UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Desarrollo de un generador procedimental de edificios

José Manuel Salas Santiago
Tutor: Carlos Aguirre Maeso

Mayo 2017

Desarrollo de un generador procedimental de edificios

AUTOR: José Manuel Salas Santiago
TUTOR: Carlos Aguirre Maeso

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo 2017

Resumen

Este Trabajo Fin de Grado pretende desarrollar una herramienta que permita generar modelos 3D de edificios con la mayor precisión posible. Para generar dichos edificios se utilizará generación procedimental por lo que la base del proyecto es la elaboración de un algoritmo que permita generar distintos modelos de edificios a partir de los parámetros que reciba por parte del usuario.

Se pretende que la herramienta se utilice sobre la plataforma web por lo que el trabajo se ha desarrollado en lenguaje JavaScript utilizando la librería de código abierto Three.js para obtener funcionalidades básicas en la generación de modelos 3D.

El desarrollo de la librería JavaScript que permita realizar la funcionalidad comentada se ha desarrollado de forma modular para facilitar la ampliación de funcionalidad y depuración de código.

Se han investigado los algoritmos de proyectos realizados anteriormente que buscaban la generación de edificios o ciudades de forma procedimental para recoger las características que todos consideraban imprescindibles y aquellas destacaban por aumentar el rendimiento o minimizar los errores.

Se ha decidido proporcionar tres modos de generación del edificio, uno por defecto que se utilizaría para comprobar la integración de la librería, uno aleatorio que podría ofrecer ideas de diseño al usuario y permite ver el alcance del algoritmo a la hora de generar modelos y por último el modelo paramétrico en el que el usuario indica el número de parámetros que desee para hacer su modelo más o menos específico.

Dentro de lo anterior, a la hora de generar la estructura de los edificios se ha priorizado que los modelos sean lo más realistas posibles y se han dejado a parte las construcciones contemporáneas poco habituales como edificios con una base de menor tamaño que el resto del edificio.

Hay que señalar también que para la generación de los modelos de elementos decorativos se ha utilizado la herramienta Blender de forma que el usuario pueda también generar sus propios elementos decorativos y usarlos en sus edificios.

Por último, además del desarrollo del algoritmo se ha diseñado una sencilla interfaz de usuario que permita comprobar visualmente los resultados que ofrece el trabajo realizado además de ejemplificar como se añadiría y haría uso de la librería implementada en el proyecto de un usuario externo.

Abstract

This Bachelor Thesis aims to develop a tool that allows generate 3D models of buildings as accurately as possible. To generate such buildings, it will be used procedural generation, reason why the base of the project is the elaboration of an algorithm that allows to generate different models of buildings from the parameters that receives by the user.

It is intended that the tool be used on the web platform so the work has been developed in JavaScript language using the open source library Three.js to get basic functionality in the generation of 3D models.

The development of the JavaScript library that allows to realize the commented functionality has been developed in a modular way to facilitate the extension of functionality and code debugging.

I have investigated the algorithms of previous projects that looked for the generation of buildings or cities in a procedural way to collect the characteristics that all considered essential and those that stood out by increasing the performance or to minimize the errors.

I have decided to provide three modes of building generation, one by default that would be used to check the integration of the library, a random one that could offer design ideas to the user and allows to see the scope of the algorithm when generating models and finally the parametric model in which the user indicates the number of parameters he wants to make his model more or less specific.

In the foregoing, at the time of generating the structure of the buildings, it has been prioritized that the models are as realistic as possible and have left aside unusual contemporary constructions such as buildings with a smaller base than the rest of the building.

It should also be noted that for the generation of decorative elements models the Blender tool has been used so that the user can also generate their own decorative elements and use them in their buildings.

Finally, in addition to the development of the algorithm, a simple user interface has been designed to visually check the results of the work performed, as well as to illustrate how the implemented library would be added and used in the project of an external user.

Palabras clave

Generación procedimental, malla de polígonos, polígono, geometría, material, textura, modelo 3D, librería, vértice, generar, aleatorio, caja limitada, algoritmo, edificio, planta, azotea.

Keywords

Procedural generation, polygonal mesh, polygon, geometry, material, texture, 3D model, library, vertex, render, random, bounding box, algorithm, building, floor, rooftop.

Agradecimientos

En primer lugar, me gustaría agradecer a todos los profesores que se han preocupado por nuestro aprendizaje a lo largo de la carrera y nos han permitido completar nuestra formación. Un agradecimiento especial a Carlos Aguirre por darme la oportunidad de trabajar en un tema tan interesante como la generación procedimental y que como tutor de este Trabajo Fin de Grado ha reconocido mi esfuerzo y me ha ayudado a corregir y mejorar la entrega final.

También quiero agradecer a mis compañeros de la carrera el hacer más sencilla, interesante y entretenida la experiencia de formarnos en esta materia.

Dar gracias a mis amigos por ofrecer siempre momentos para desconectar, relajarnos y divertirnos y permitirme recordar siempre lo que es más importante en la vida.

Por último, doy gracias a mi familia por apoyarme siempre y ser como son, en especial agradecer a mis padres el haberme permitido seguir estudiando y formarme sin tener que preocuparme de otras cosas y el no haberme agobiado nunca dejando que me organizase como yo viera oportuno.

ÍNDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

La generación procedimental o por procedimientos de objetos es una técnica que se ha utilizado a lo largo de la historia, pero cuyo uso se ha extendido y afianzado en los últimos años. Esta técnica es utilizada para la creación de diferentes contenidos como podrían ser diseños de niveles en un videojuego, personajes o elementos decorativos, los eventos que ocurren en su historia o incluso las texturas que simulen superficies o materiales.

Utilizar esta técnica permite abaratar costes de desarrollo y distribución permitiendo que la cantidad de personas que se requieren para desarrollar videojuegos se vea reducida dando lugar a grupos independientes de pocas personas que son capaces de producir sus propios videojuegos en espacios de tiempo más reducidos.

Algunos ejemplos del uso de esta técnica son: .kkrieger desarrollado por .theprodukkt un grupo de 6 personas cuya beta fue lanzada en 2004 y que destaca por su tamaño; tan solo 96 KB que contienen un completo juego FPS que hace un uso intensivo de la generación por procedimientos. En el mismo contexto podemos nombrar a Minecarft desarrollado al comienzo por una única persona o Terraria desarrollado por tres personas, ambos son un ejemplo de mundos que se regeneran cada partida permitiendo una experiencia de juego infinita. Hay que nombrar también al actual No Man´s Sky [1] dirigido a la generación procedimental de planetas y galaxias.

Las principales ventajas que nos ofrece la generación procedimental son obtener archivos de menor tamaño para nuestro proyecto, aumentar el número de elementos que podemos crear de forma manual o incluyendo reglas aleatorias que nos generen elementos nuevos que puedan ser utilizados y crear una experiencia más aleatoria al poder obtener entornos diferentes que cumplan con las limitaciones que requerimos mediante el diseño de un algoritmo.

La generación de entornos urbanos es una tarea muy común en la actualidad y que de forma manual conlleva un coste temporal muy elevado. Por ello, para realizar esta tarea se suele utilizar la técnica de generación procedimental.

En nuestro caso para el diseño de edificios la diferencia entre almacenar un gran conjunto de edificios totalmente personalizados de forma manual o una serie de elementos que nos permiten generar de forma procedimental un edificio con el uso de un algoritmo y una serie de parámetros que ofrecen esa personalización de los diseños manuales desemboca en un gran ahorro de espacio y tiempo en el desarrollo de un proyecto.

1.2 Objetivos

El objetivo de este TFG es el desarrollo de una librería en lenguaje JavaScript que con el uso de Three.js [2] y WebGL permita generar edificios personalizados a través de parámetros. De esta forma el proyecto ofrecerá una herramienta de generación de edificios elaborada con código abierto.

La librería permitirá la creación de modelos 3D de edificios realistas a través de un algoritmo que se basará en la idea de la generación procedimental tanto para el diseño de la estructura como de las texturas del edificio, permitiendo que el modelo sea generado a través de unos pocos parámetros y que se pueda personalizar el máximo posible.

Además, se implementará un simulador con el que el usuario podrá comprobar la versión final del producto, dicho simulador será útil también como herramienta de depuración para el desarrollo del proyecto principal del cual dicho simulador no forma parte.

1.3 Organización de la memoria

La memoria está compuesta de 7 capítulos y X anexos, los cuáles se enumeran a continuación:

• **Capítulos:** Secciones del documento donde se detallan todas las características del trabajo realizado, desde el estudio de proyectos similares, al diseño y desarrollo del algoritmo propio para la implementación de la librería.

- 1-**Introducción:** presente sección en la cual se da un primer vistazo al objetivo del TFG y la organización de la presente memoria.

- 2-**Estado del arte:** en esta sección se hará una mención a las técnicas de generación procedimental, productos comerciales que las utilizan y algunos ejemplos de proyectos realizados dirigidos a la generación procedimental de entornos urbanos y edificios.

- 3-**Analisis:** en esta sección se detallan requisitos funcionales y no funcionales que deberá cumplir la librería para tener un rendimiento adecuado y un realismo suficiente a la hora de elaborar modelos de edificios.

- 4-**Diseño:** en esta sección se detallarán aspectos del diseño principal del modelo de los edificios, su forma de construirse, tecnologías utilizados para llevar a cabo el modelado y la funcionalidad general de cada parte de la librería.

- 5-**Desarrollo:** en este capítulo se analizará en detalle la implementación del algoritmo, decisiones de diseño específicas y características destacables de la librería.

- 6-**Integración, pruebas y resultados:** en esta sección se detallarán las diferentes pruebas realizadas sobre la librería y los resultados que se han obtenido para concluir que se ha conseguido el objetivo propuesto.

- 7-**Conclusiones y Trabajo futuro:** en este capítulo se comentarán las conclusiones obtenidas del trabajo realizado y posibles mejoras o ampliaciones de funcionalidad que se podrían realizar sobre la librería.

2 Estado del arte

La generación por procedimientos de contenido para videojuegos es una técnica que como hemos visto se ha utilizado en el pasado, sin embargo, no existe una gran cantidad de motores de generación de contenido de este tipo abiertos al público, dicho número se reduce aún más si buscamos una herramienta que sea gratuita y para una plataforma concreta.

Algunos motores multiplataforma como Unity ofrecen este tipo de herramientas con los inconvenientes de ser código cerrado y formar parte de dicho motor comercial, es por ello que el desarrollador puede verse limitado si su diseño es algo poco común o complejo ya que este tipo de herramientas suelen tender a ofrecer resultados realistas.

Es por esto que he realizado una búsqueda sobre las diferentes técnicas de modelado por procedimientos utilizadas y algunos proyectos que las hayan utilizado para poder plantear una base sobre la que construir mi algoritmo.

2.1 Técnicas de modelado procedimental

La generación de contenido de forma procedimental puede usarse de dos formas; generando el contenido en tiempo real o generarlo previamente a la ejecución. Usando la segunda técnica podemos revisar los resultados para comprobar que son válidos, esto nos dará la seguridad de que todos los usuarios interactúan en un entorno similar, pero nos penalizará en el almacenamiento de los resultados completos.

A continuación, se explicarán algunas de las técnicas utilizadas para el modelado procedimental y algunos ejemplos de proyectos que han hecho uso de ellas y sirven de antecedente para este proyecto.

2.1.1 Fractales

Los fractales fueron introducidos por Benoît Mandelbrot en 1975, el nombre de fractal viene del latín fractos; roto o fracturado. Un fractal es conocido como un objeto geométrico con una estructura básica que se repite a diferentes escalas. Muchas estructuras naturales son de tipo fractal, aunque no eran conocidos por este nombre este tipo de estructuras ya se conocían en la matemática a principios del siglo XX [3].

Un objeto geométrico fractal es autosimilar es decir está formado por copias más pequeñas de la misma figura. Es por ello que puede definirse con un algoritmo recursivo. [3]

En esta figura podemos ver el ejemplo de un objeto fractal; la alfombra de Sierpinski [4]

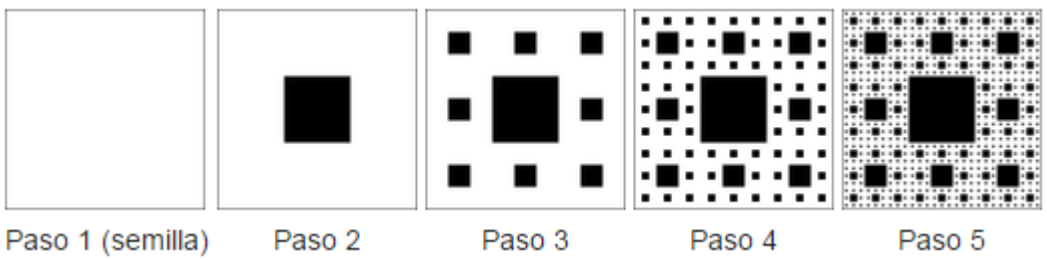

Figura 2-1: La alfombra de Sierpinski

2.1.2 Sistema de Lindenmayer

La técnica de sistema de Lindemayer o sistema-L se basa en un conjunto de reglas que permiten el modelado de la morfología de un organismo. Cuando Aristid Lindenmayer introdujo la técnica lo utilizó para definir la relación entre las células de las plantas, posteriormente lo usaría para describir patrones de crecimiento de plantas complejas. [5]

"La naturaleza recursiva de las reglas de los sistemas-L conduce a la autosemejanza y por tanto facilita la descripción de formas tipo fractales." [5]

Los sistemas-L pueden definirse como un conjunto (V, S, W, P) donde V es el conjunto de elementos que pueden ser reemplazados, S el conjunto de elementos que se mantienen fijos, W un conjunto de elementos de V que permiten definir el axioma del sistema y P el conjunto de reglas que indican la serie de elementos de V que podremos reemplazar por combinaciones de elementos de V y S. [5]

Al aplicar las reglas del conjunto P sobre el axioma del sistema de forma iterativa so obtienen estructuras complejas. Algunos ejemplos de aplicar estos sistemas serían modelar el crecimiento de las algas o la secuencia de números de Fibonacci.

Este tipo de técnica es más útil para estructuras de tamaño indefinido como plantas o mapas de carreteras en lugar de elementos con un tamaño máximo acotado como es el caso de los edificios.

A continuación, puede verse un ejemplo de aplicación de sistema-L; el triángulo de Sierpinski:

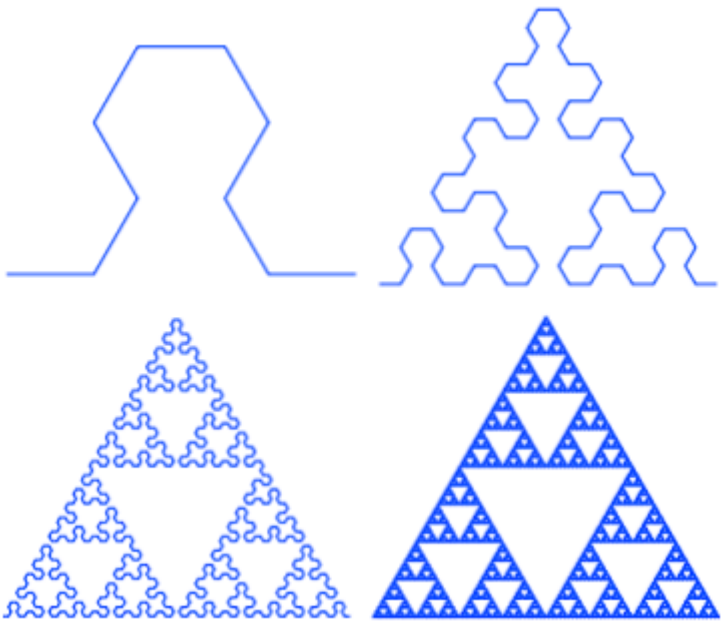

Figura 2-2: Derivaciones del Triángulo de Sierpinski

2.2 Antecedentes

En este apartado aparecen algunos proyectos que han utilizado las técnicas vistas para la construcción de entornos urbanos y edificios a través de algoritmos.

2.2.1 Procedural Modeling of Cities

Proyecto dirigido por Yoav Parish y Pascal Müller, con el uso de sistemas-L crearon un algoritmo aplicado a la generación de ciudades, que se encargaba tanto del diseño de los edificios como el mapeo de las calles. [6]

Para elaborar los edificios establecen un espacio acotado donde se construirá la base y posteriormente se aplican las reglas de modelado según el espacio disponible para ese edificio en particular. En la siguiente figura puede verse la generación de un edificio a lo largo de 5 pasos:

Figura 2-3: Generación de edificio en 5 pasos

2.2.2 CGA Shape

Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer y Luc van Gool presentaron un nuevo método para la generación de edificios de forma procedimental basándose en trabajos anteriores de generación de ciudades, pero añadiendo nuevas reglas que aumentaban el realismo de los edificios.

Las reglas de CGA Shape se introducen por parte del usuario y pueden realizar procedimientos como redimensionar, rotar o mover la geometría según la situación, añadir nuevos elementos al edificio o reducirlo según el espacio por el que esté acotado. [7]

Cada una de las reglas que se pueden aplicar tienen unas condiciones para cumplirse además de un factor de aleatoriedad para elegir entre varias reglas en caso de hubiera múltiples posibilidades. Se usa el mismo sistema tanto para las geometrías 3D o 2D, al tener tanta capacidad de modificación esto les permite generar tanto un entorno actual con edificios de oficinas como entornos antiguos como es el caso de una simulación de la ciudad de Pompeya que podemos ver en la figura 2-4.

Figura 2-4: Varias vistas del modelo de la ciudad de Pompeya

2.2.3 Procedural House Generation

Existen varios proyectos que se centran en la generación del edificio a partir del diseño del plano con la separación de las distintas habitaciones. En el caso del algoritmo propuesto por Jess Martin [8] tenemos un árbol de nodos en el cual el nodo raíz es aquel que tiene la puerta de salida a la calle y los demás nodos hijos son las habitaciones que se comunican con el nodo raíz o aquellas que se comunican con otra habitación que se comunica con la raíz y así de forma recursiva.

Se da más prioridad a unas habitaciones que a otras de forma que al ir creándose las habitaciones cuando se toquen las paredes en extensión de dos habitaciones la que tenga más prioridad se siga expandiendo hasta que el conjunto ocupe el espacio acotado al edificio. Según pruebas realizadas el algoritmo es capaz de generar 50.000 casas en menos de 3 minutos [8].

Aquí se puede ver un ejemplo en el que tras generar el plano se ha construido el modelo del edificio:

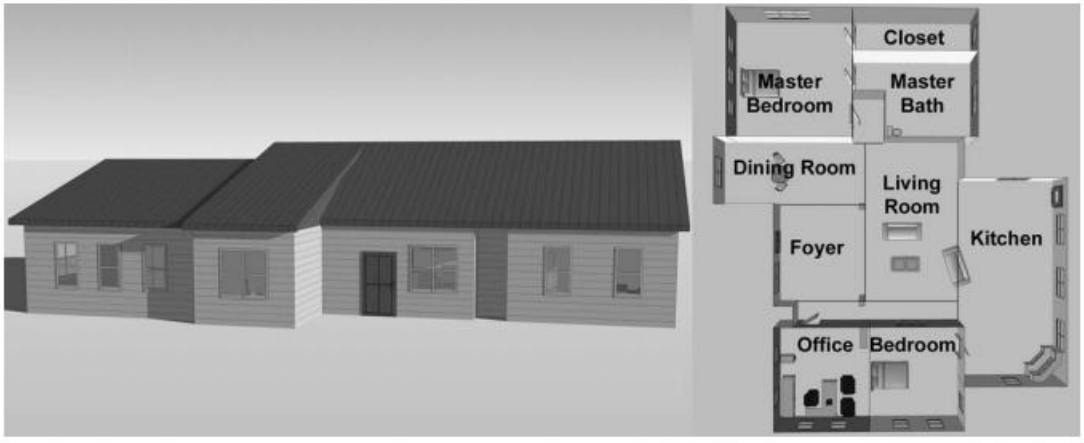

Figura 2-5: Casa generada a partir del plano interior producido por el algoritmo de Martin

2.2.4 Citygen

Citygen es un proyecto dirigido por George Kelly y Hugh McCabe, este proyecto está dirigido a la construcción de ciudades, pero me centraré únicamente en la construcción de edificios. Según exponen en su artículo en primer lugar hacen un proceso de división de zonas basado en el terreno que tienen disponible limitado por el mapa de carreteras de la ciudad, una vez tienen las divisiones en vecindarios o manzanas comienzan a hacer subdivisiones para los edificios de esa zona específica los cuáles eligen basándose en el espacio que tienen disponible. Distinguen, por ejemplo, entre edificios de oficinas que están más destinados a aprovechar todo el espacio disponible y casas con jardín que necesitan espacio extra y que la casa esté alejada de la acera. [14]

Comienzan construyendo los edificios desde la base como en el caso de otros algoritmos y construyen hacia arriba, utilizan así mismo texturas únicas para las ventanas que se usan de forma repetida a lo largo del edificio. Como mencionan el resultado final del edificio no es excesivamente realista pues el algoritmo está destinado a generar ciudades, dicho resultado se puede ver en la figura 2-6. [14]

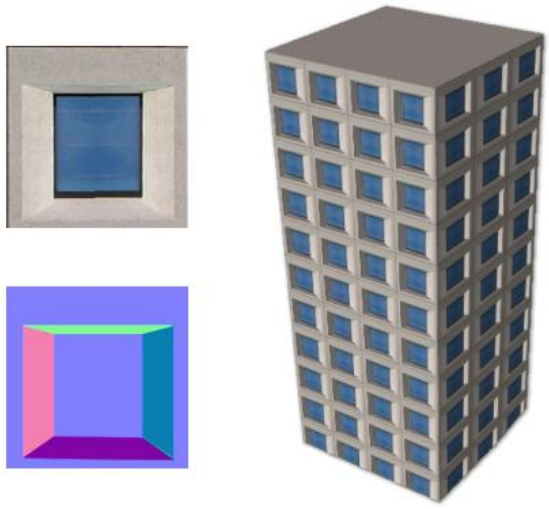

Figura 2-6: Edificio simple generado de forma procedimental

2.2.5 A Survey of Procedural Techniques for City Generation

En este artículo de nuevo de George Kelly y Hugh McCabe nos comparan las diferentes técnicas utilizadas para la generación procedimental de edificios y ciudades, comparan proyectos de su autoría como el anteriormente reseñado y proyectos de otras personas, para cada uno de ellos buscan responder si se cumplen varias propiedades que deberían conseguir superar nuestros modelos generados. [15]

Las propiedades más interesantes de las que hablan son:

Realismo, comparan los resultados en este aspecto tanto en el factor aleatorio de los diferentes edificios que se generan en las ciudades como el resultado singular de un edificio, comprobando si el modelo se asemeja suficiente a un edificio real.

Variabilidad, de nuevo se centran en la diferencia entre cada edificio, las posibilidades que proporciona cada técnica para permitirnos alterar en mayor medida los modelos de los edificios de forma que solo podamos obtener dos modelos muy diferentes o dos casi idénticos que cambien en algún pequeño factor.

Valores de entrada, es decir, parámetros que tiene que enviar el usuario al algoritmo, en casos como el algoritmo que he implementado existen modos de generación que solo requieren que el usuario indique que ese es el modo que quiere usar, mientras que como en el caso de muchos algoritmos que mencionan es necesario introducir una cantidad moderada de parámetros.

Eficiencia y tiempo real, referido a si es necesario tener modelos previos o se generan completamente en tiempo real como es el caso de mi algoritmo y a la eficiencia de generación relacionada con la complejidad del modelo a generar.

He tenido en cuenta las diferentes comparaciones expuestas en el artículo tanto para elegir la técnica más apropiada para el proyecto como a los aspectos que más debía cuidar a la hora de desarrollar el algoritmo.

2.2.6 Automatic Generation of 3D Building Models with Efficient Solar Photovoltaic Generation

En 2016 Kenichi Sugihara y Zhenjiang Shen publicaron un artículo muy interesante relacionado con la generación de modelos 3D para edificios, en este caso buscando una forma de diseñar ciudades en la que se puedan introducir paneles solares en los tejados de las casas. Como en mi caso hicieron una investigación del trabajo plasmado en otros artículos mencionando varios de los artículos reseñados en esta sección. Comentan que la generación de modelos 3D para un edificio con una herramienta dedicada es un proceso excesivamente costoso en términos de tiempo de trabajo, por lo que orientaron su proyecto a la generación automático de dichos modelos. [16]

Para generar sus modelos en primer lugar crean las formas más básicas como cubos o prismas, después añaden las ventana, puertas y demás elementos decorativos, una vez añadidas se realizan rotaciones en los elementos para obtener un modelo consistente. [16]

Una vez hecho esto construyen la segunda planta de la casa, tras situar los elementos de dicha planta se texturiza el modelo y en caso de que el edificio tenga más de dos plantas se repite la configuración de la segunda planta en los pisos superiores. [16]

Por último, como ya he mencionado su proyecto va destinado a generar edificios que puedan albergar paneles solares por lo que la forma de los tejados tiene un papel especial. Como con el resto del edificio ellos eligen dividir la geometría en rectángulos en lugar de triángulos, con esto dividen el tejado en dos fracciones e inclinan dichas partes según la cantidad de energía que requiere la casa, en pocas palabras estiman que grado de inclinación requiere el techo para que el panel solar reciba en un grado u otro la luz solar y genere más o menos energía. [16]

Figura 2-7: Modelos de edificios con tejado partido para orientarse a la luz solar

3 Análisis

En este apartado se especificarán las características que debe cumplir la librería para poder generar los edificios con el resultado esperado, para ello se especificarán los requisitos tanto funcionales como no funcionales que deberá cumplir el proyecto.

3.1 Requisitos funcionales

Comportamiento del algoritmo y salidas que se espera obtener según las entradas enviadas a la herramienta como parámetros.

En este apartado me centro en especificar los aspectos que debe cumplir el proyecto por una parte en lo referente a que cada edificio tenga unos valores de carácter aleatorio que lo diferencien de todos los demás pero que estén controlados para no generar un modelo de edificio inconsistente y por otra a la cantidad de personalización que se debe ofrecer al usuario para que pueda obtener el modelo más ajustado a lo que él desea.

Código	Requisito
RF-01	Aleatoriedad: el algoritmo será capaz de generar edificios distintos para los mismos parámetros siempre cumpliendo con el nivel de realismo esperado. De esta forma cualquier parámetro que no sea especificado se obtendrá de forma aleatoria para aumentar la variedad, en algunos casos en lugar de un valor aleatorio será un valor por defecto ya que dicho valor puede ser vital para la consistencia.
RF-02	Consistencia: los edificios generados por el algoritmo deben tener sentido estructural de forma que durante la generación procedimental no haya intersección de pareces, ventanas que se superpongan, etc. Además, no se permitirá que las plantas superiores sean de mayor tamaño que la base, aunque esto si se puede ver en algunos diseños reales no es algo común.
RF-03	Personalización: la librería ofrecerá gran cantidad de parámetros para personalizar al máximo la estructura del edificio, aunque no todos deberán ser especificados como se indicó anteriormente. Algunos de estos parámetros son la separación de ventanas, la agrupación de ventanas, la altura de las plantas, etc.
RF-04	Personalización: la librería permitirá especificar las texturas que se utilizarán para la fachada, el techo, los balcones, etc. Dichas texturas deberán ser compatibles con lo requerido por el algoritmo, es decir deben respetar un patrón que permita repetirlas horizontalmente a lo largo de la fachada sin resultar irreales.
RF-05	Personalización: la librería ofrecerá varios modos de generación de edificios, los cuáles son: modo por defecto que devuelve un edificio de muestra para que el usuario compruebe la capacidad del algoritmo, aleatorio en el que todos los parámetros para generar el edificio se obtendrán de forma aleatorio dando un resultado consistente y modo paramétrico en el que se especificará una serie de parámetros para acotar el resultado esperado.

Tabla 3-1: Requisitos funcionales de la librería de generación de edificios

3.2 Requisitos no funcionales

Requisitos no funcionales que deberá cumplir la librería de generación de edificios.

En este caso los aspectos que trato son características generales que debería tener cualquier aplicación: que la librería sea sencilla de usar y exista una herramienta; manual, para que el usuario pueda aprender a usarla sin conocimiento previo, que el código sea portable fácil de ampliar y que la librería ofrezca un rendimiento aceptable, es decir, permita diseñar modelos de forma sencilla y generarlos en poco tiempo.

Código	Requisito
RNF-01	Usabilidad: uso sencillo de la librería permitiendo que el usuario no necesite especificar todos los parámetros para la generación del edificio.
RNF-02	Usabilidad: existencia de un manual de usuario que explique el uso de la librería y que se encuentra en la presente memoria.
RNF-03	Portabilidad: dado que el código externo que utiliza la librería es código abierto puede hacerse uso de la librería en cualquier proyecto con solo incluir las librerías necesarias.
RNF-04	Extensibilidad: la librería estará diseñada con un código claro y estructurado que permita incluir nuevas funcionalidades de personalización en el futuro.
RNF-05	Rendimiento: la librería deberá ofrecer la generación de un edificio sencillo de tamaño estándar en no más de un segundo y será capaz de ofrecer un edificio de gran tamaño y mayor complejidad en unos pocos segundos.

Tabla 3-2: Requisitos no funcionales de la librería de generación de edificios

4 Diseño

El código del proyecto se ha desarrollado de forma modularizada para poder desarrollar con mayor facilidad y permitir la ampliación de funcionalidades con mayor velocidad.

Para construir el modelo del edificio en primer lugar se parsean los parámetros que se hayan recibido y en función de ellos se van generando las plantas fusionando su geometría con la del edificio, a su vez en la generación de cada planta en primer lugar se construye el modelo básico y después se añaden sobre él los elementos que se quiera por ejemplo las puertas en la primera planta, los balcones en una planta intermedia y elementos del tejado sobre la última planta.

Esto permite que si por ejemplo se quisieran añadir jardines en la primera planta podría hacerse añadiendo el código en la creación de plantas y usándolo al generar la planta base. Además, si se quisieran añadir modelos externos a los utilizados por la librería para los balcones y elementos del tejado solo habría que modelarlos en una herramienta externa, utilizando también una textura externa e incluir el nuevo elemento en los parámetros del edificio.

En mi caso para elaborar los modelos de los elementos del edificio como balcones y puertas he utilizado la herramienta Blender [10] que nos permite construir modelos totalmente personalizados vértice a vértice y mapear sobre el modelo final una textura personalizada que se ajuste correctamente como veremos más adelante.

Como se ha mencionado anteriormente el proyecto se ha realizado apoyándose sobre la librería de código abierto Three.js que nos facilita algunos aspectos de la generación de los edificios.

4.1 Three.js

Three.js es una biblioteca de javascript desarrollada por Ricardo Cabello cuyo código abierto puede encontrarse en la web, posee una gran cantidad de documentación además de una comunidad activada de desarrolladores y usuarios por lo que se puede encontrar gran variedad de ejemplos.

Existen varias técnicas para el renderizado del edificio, a continuación, se comentarán tres de ellas incluida la elegida para este proyecto; malla de polígonos.

- **Voxels:** un voxel es una unidad cúbica que forma parte de un objeto tridimensional. Constituye la unidad mínima que puede procesar una matriz dimensional lo cual equivaldría a un pixel en un modelo 2D [11]. Un ejemplo de videojuego que utiliza esta técnica de renderizado es el famoso Minecraft que ya fue mencionado anteriormente por utilizar la generación procedimental. Aquí puede verse un ejemplo del modelo de un objeto generado a partir de voxels.

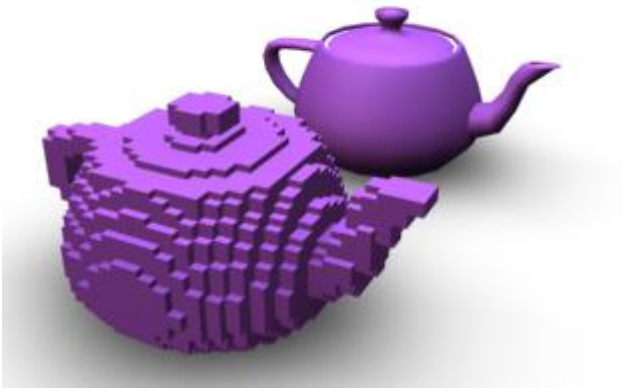

Figura 4-1: Modelo de objeto generado con voxels

• **Nube de puntos:** una nube de puntos es un conjunto de vértices sobre un sistema tridimensional, pueden obtenerse nubes de puntos de un objeto a través de un escáner 3D o por fotogrametría digital para obtener modelos tan realistas como los que aparecen en la figura 3-2. Una nube de puntos puede ser desde el esquema de un objeto hasta una imagen con resolución de calidad fotográfica ya que los puntos que componen la nube son en última instancia píxeles.

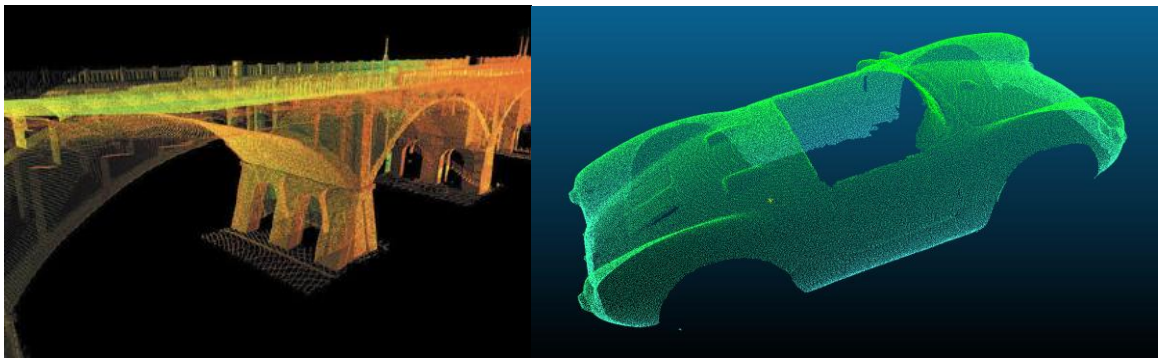

Figura 4-2: Nubes de puntos de un puente y el chasis de un coche

• **Malla de polígonos:** una malla poligonal o de polígonos (mesh en inglés) es una Superficie creada con un método tridimensional generado por conjuntos de vértices en un espacio virtual. Una malla poligonal se construye a partir de caras de 3 vértices que se considera la unidad básica de un polígono tridimensional. [12]

Crear dichas caras de tres vértices es una técnica conocida como triangulación que se utiliza en la librería implementada y se comentará más adelante. Esta técnica tiene problemas de rendimiento ya que convertir el modelo de un objeto a una malla de polígonos puede ser complicado de implementar y provocar un cose computacional alto.

Para resolver estos problemas de rendimiento optimizaremos los elementos de la malla haciendo que los conjuntos de triángulos formen tiras o abanicos de triángulos que permiten que la GPU procese dichos polígonos de forma más compacta y rápida. Esta mejora se produce porque los triángulos compartirán vértices que podrán ser descartados, esto se realizará en la implementación de la librería.

A continuación, se puede ver una figura de un objeto renderizado usando la técnica de malla de polígonos.

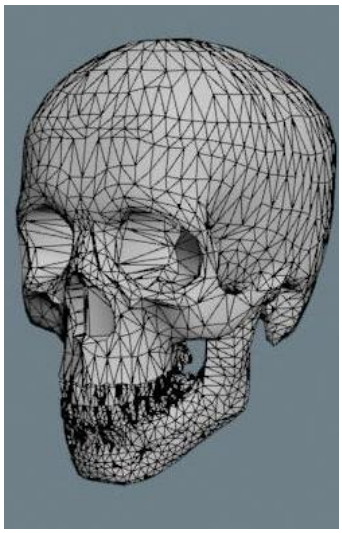

Figura 4-3: Representación de una calavera con la técnica de malla poligonal

Aquí se puede ver un ejemplo de abanico de triángulos donde varios vértices podrían descartarse para mejorar el rendimiento.

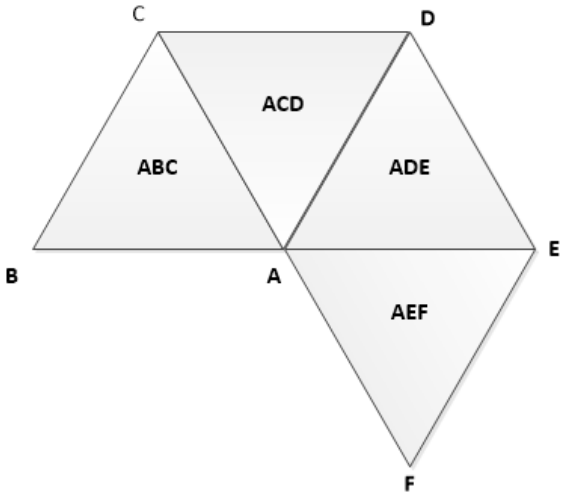

Figura 4-4: Abanico de triángulos

A continuación, se enumerarán las diferentes utilidades de la librería Three.js que se han usado en la implementación del proyecto. Para poder utilizarlas correctamente se estudió su funcionalidad en la documentación de la librería [9] que se ha utilizado también para indicar dicha funcionalidad en esta sección.

- **Vector2:** par de números que se utiliza para representar un punto en un espacio 2D.
- **Vector3:** conjunto de 3 números utilizado para representar un punto en un espacio 3D.
- **Geometry:** clase básica para todas las geometrías, utilizada para la creación de la geometría base del edificio y para la creación de la geometría de las diferentes plantas que se van fusionando sobre el edificio.

- **Face3:** Superficie triangular compuesta por tres vértices, superficie mínima utilizada para crear la estructura de cada planta y del edificio completo.
- **ShapeUtils.triangulateShape:** función que nos crea un array con grupos de tres vértices que utilizamos para crear superficies triangulares que en conjunto cubran una superficie irregular, esta funcionalidad se utiliza para la construcción de las azoteas cuyas formas pueden ser irregulares.
- **Geometry.applyMatrix:** función que realiza una transformación sobre la geometría según una Matrix4.
- **Matrix4.makeTranslation:** función que aplica un desplazamiento sobre la matriz 4x4, dicha matriz se aplica sobre una geometría con la función anterior y nos permite situar correctamente las plantas, el edificio, las ventanas y demás elementos de la geometría.
- **Euler:** clase que representa ángulos de Euler para describir rotaciones de una geometría.
- **Matrix4.makeRotationFromEuler:** función que usando ángulos de Euler aplica una rotación sobre la matriz 4x4, dicha matriz se aplica sobre una geometría con la función applyMatrix que nos permite rotar los elementos como balcones y puertas para situarlos en la orientación correcta.
- **ColladaLoader:** proyecto desarrollado sobre la librería Three.js que nos permite cargar los modelos de los elementos del edificio que desarrollamos con la herramienta Blender.
- **Mesh:** clase que representa una malla poligonal de triángulos, se construye a partir de una geometría y un material.
- **MultiMaterial:** es un material utilizado para especificar varios materiales en la misma geometría, la cual elige que material utilizar para cada cara según los índices de los arrays de materiales y caras.
- **TextureLoader:** función de three.js que nos permite cargar las texturas para las diferentes partes del edificio.
- **RepeatWrapping:** uno de los modos de mapear la textura de forma horizontal, en este caso como hemos diseñado las texturas para que sean regulares y puedan repetirse se elige esta opción.
- **MeshPhongMaterial:** material utilizado para las geometrías del edificio, ofrece un mayor realismo a la representación a costa de algo de rendimiento.
- **DoubleSide:** indica que el material del modelo sea visible desde ambas partes de la cara.

4.2 Blender

Se ha utilizado la herramienta Blender para diseñar elementos del edificio como puertas, balcones o antenas que se sitúan en las azoteas. Para construir dichos elementos necesitamos situar todos los vértices del elemento a nuestro gusto y posteriormente hacer un mapeado sobre el modelo para indicar como se debe renderizar la textura.

El modelo de la textura se guarda en formato .dae que es formato por defecto para cargar modelos con la herramienta ColladaLoader anteriormente mencionada.

Como se indicó anteriormente si el usuario quisiera incluir sus propios modelos para los elementos de la fachada y las azoteas podría diseñarlos con esta herramienta o una similar y enviar la localización del modelo como parámetro para que la librería pueda cargarlos y usarlo normalmente.

A continuación, se muestra el diseño de un balcón con toldo para la fachada del edificio, como se puede apreciar el modelo está compuesto de un conjunto de triángulos.

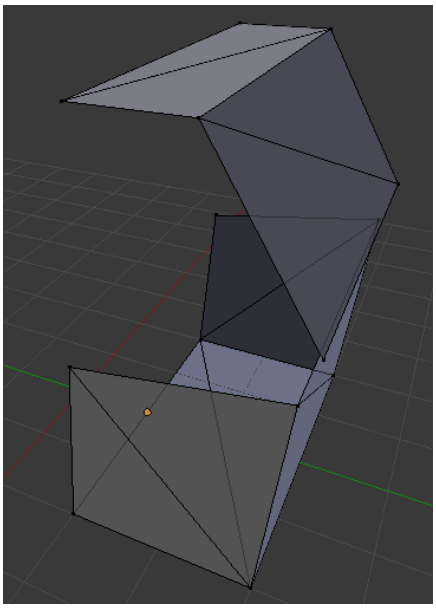

Figura 4-5: Modelo de un balcón con toldo diseñado en Blender

4.3 Librería y simulador

En esta sección se profundizará en el diseño de la librería indicando las diferentes elecciones de generación de geometría, texturización de los elementos del modelo, implementación de los algoritmos matemáticos utilizados y un diagrama de los diferentes módulos que componen la librería.

4.3.1 Geometrías y texturización

En esta sección se profundizará en el diseño de la librería. Para construir el edificio se plantean tres modos; el modo por defecto genera el modelo de un edificio simple de ejemplo que se ha utilizado para depurar en el desarrollo y puede servir de prueba para el usuario. El modo aleatorio o "random" genera edificios totalmente aleatorios desde la forma del edificio o la cantidad de hijos que tiene la estructura a las texturas de la fachada y el tejado. Por último, el modo paramétrico es el modo principal dirigido al usuario que le permite generar edificios totalmente personalizados indicando todo tipo de parámetros y que como ya se ha mencionado utiliza valores aleatorios para aquellos parámetros que no se especifiquen.

A la hora de construir el edificio se han tenido en cuenta los diferentes algoritmos vistos en la sección de antecedentes y se ha planteado comenzar construyendo una base y añadiendo plantas encima hasta construir el edificio. En algunos de los proyectos se planteaba comenzar la construcción desde la parte más alta del edificio, pero se ha creído más conveniente hacerlo desde la base ya que facilita la implementación.

De esta forma creamos una base, la cuál puede ser simple es decir formada por un único polígono (triángulo o cuadrilátero) o por una forma más compleja que incluya extensiones del edificio o hijos como se les llamará en adelante.

Se ha considerado un árbol de nodos donde el nodo raíz sería ese primer polígono y los hijos del edificio los diferentes nodos hijos del árbol, dichos hijos podrán tener también

descendientes hasta una profundidad máxima que se ha impuesto para mantener el realismo en el edificio.

Dentro de esto hay que indicar que al añadir hijos al edificio es posible que los situemos en una esquina del nodo padre por lo que habría que descartar un nodo, esto como ya se ha mencionado está controlado en la implementación de la librería.

Para la construcción de la forma de la base se ha decidido un diseño intuitivo y ordenado que permita una implementación lo más sencilla posible. En la figura 4-6 podemos ver la forma de una base sencilla formada por un único polígono cuadrilátero, para su construcción empezaríamos por el vértice más a la izquierda y abajo y construiríamos en el sentido horario el resto de vértices. Dichos vértices se almacenan en un array de puntos o vértices de forma ordenada como se ha indicado de forma que es sencillo unir los puntos y procesarlos debido a su orden.

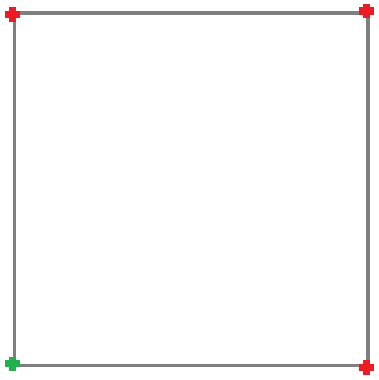

Figura 4-6: Representación de una base de edificio simple de un polígono

Si la base fuera compleja es decir formada por más de un polígono se crearían los vértices de la base y se iría iterando sobre las caras generadas, en caso de que se cumplan las condiciones para añadir un hijo en una cara, se crea y se añade de forma que si hay un hijo en la cara entre el primer y segundo vértice, los vértices irán ordenados de la siguiente manera: primer el vértice inicial de la base, luego los vértices del hijo en la primera cara, luego el segundo vértice de la base y así recursivamente en caso de haber descendientes en el hijo mencionado.

A continuación, se muestra el ejemplo de una base compuesta por 4 polígonos con los vértices numerados por su posición en el array para aclarar lo explicado anteriormente.

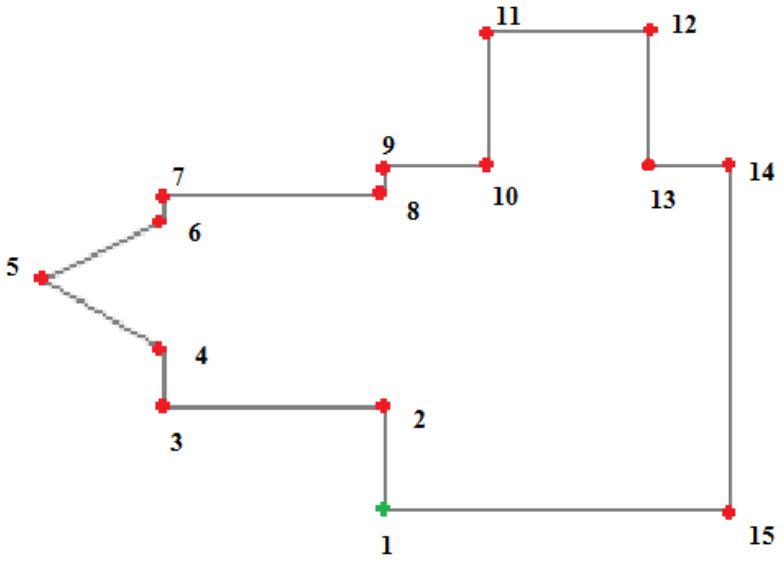

Figura 4-7: Representación de una base de edificio con cuatro polígonos

Como ya se ha mencionado los hijos pueden aparecer en diferentes formaciones, a continuación, las enumeraré con ejemplos.

Aparición en el centro de las caras del edificio, como se ve en la figura 4-8 los hijos se crean en el centro de la cara, para ello se obtiene la posición de la mitad de dicha cara y según el tamaño que vaya a tener el hijo se crea el primer vértice y el resto que formarán el hijo.

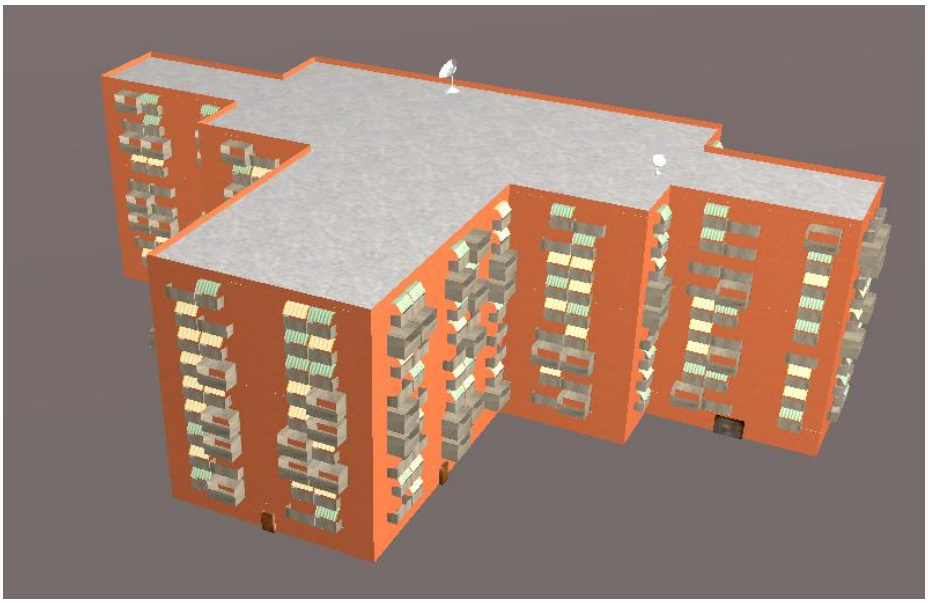

Figura 4-8: Modelo con hijos en aparición central

Aparición en fila, en este caso aparecen dos hijos en la cara donde se vayan a añadir, el primer hijo tiene su centro a un cuarto de la distancia del vértice izquierdo y derecho de la

cara y el segundo hijo a tres cuartos de dicha distancia. En este tipo de aparición se podrían añadir variante para que en lugar de dos hijos fueran más.

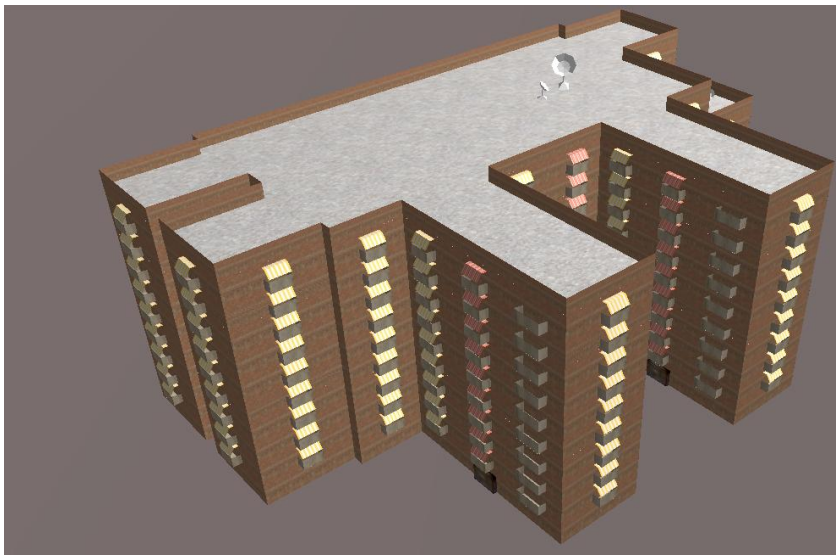

Figura 4-9: Modelo con hijos en aparición en fila

Aparición en L a la izquierda, en este caso el primer vértice del hijo coincide con el primero del padre.

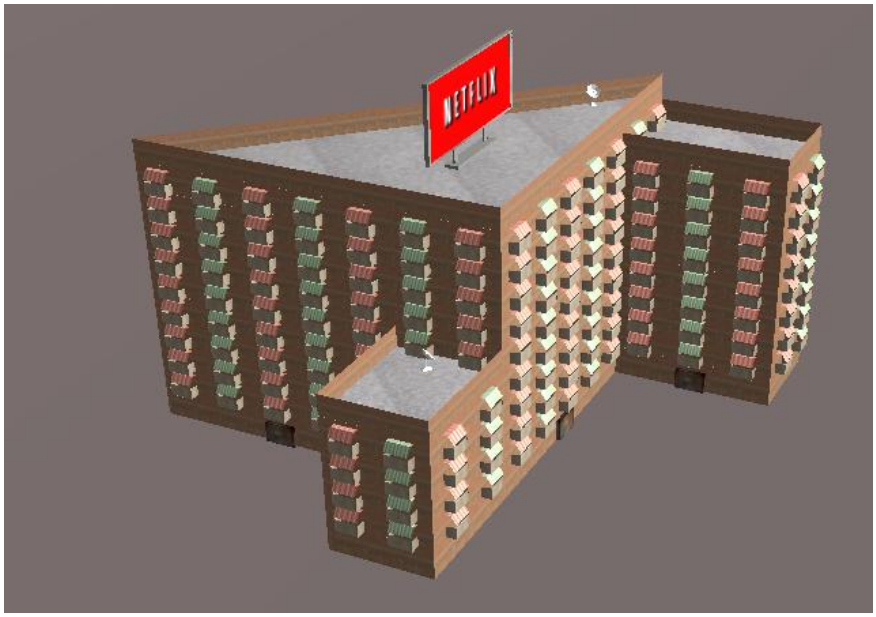

Figura 4-10: Modelo con hijos en aparición en L a la izquierda

Aparición en L a la derecha, en este caso el último vértice del hijo coincide con el último del padre.

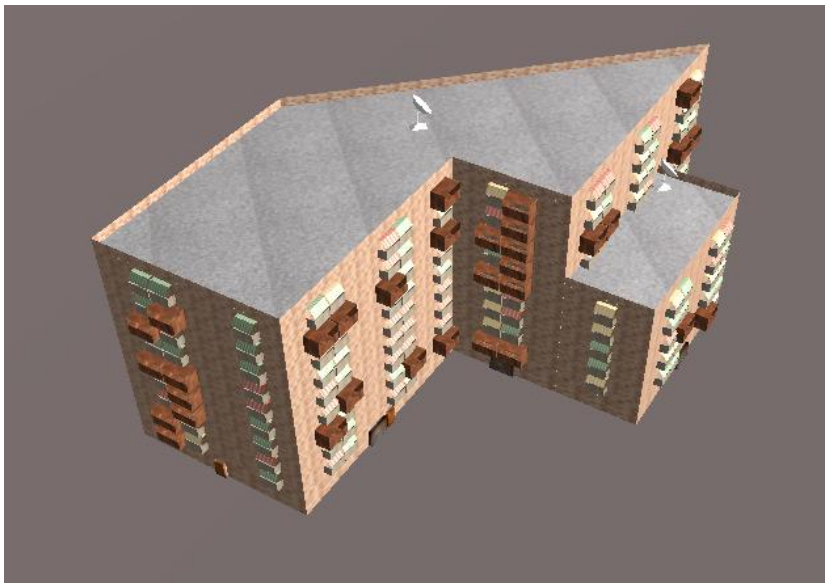

Figura 4-11: Modelo con hijos en aparición en L a la derecha

Aparición simétrica, en este caso los hijos en caso de crearse se situarán todos en la misma posición de sus respectivas caras; en la zona de la esquina izquierda de forma que el primer vértice del hijo sea el mismo que el primer vértice del padre y en la zona de la esquina derecha de forma que el último vértice del hijo sea el del padre, esta aparición puede verse como una combinación de la aparición en L a la derecha y a la izquierda.

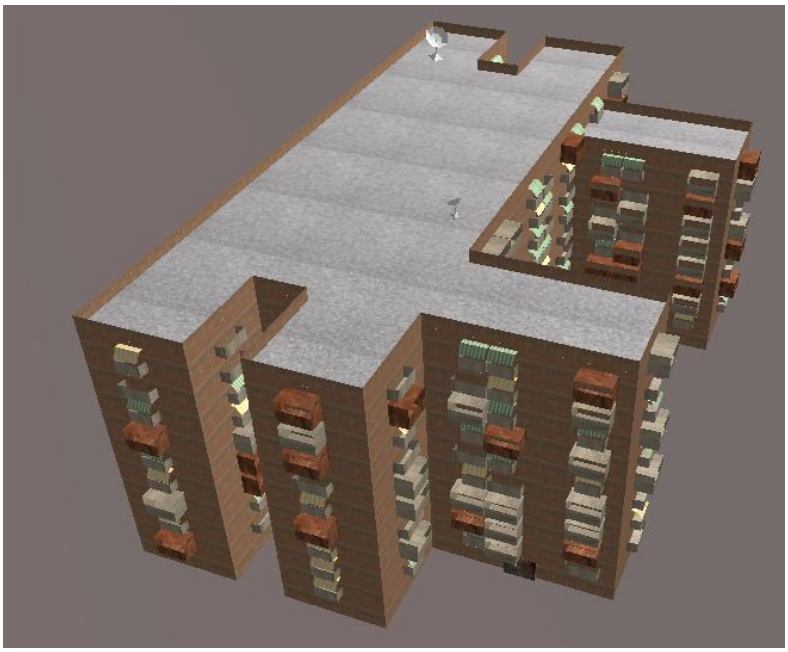

Figura 4-12: Modelo con hijos en aparición simétrica

Tanto para el caso de la aparición simétrica como a la izquierda y derecha ha sido necesario calcular la posición exacta donde colocar los centros de los hijos ya que si lo situáramos en las esquinas directamente resultaría un modelo poco realista como el de la siguiente figura.

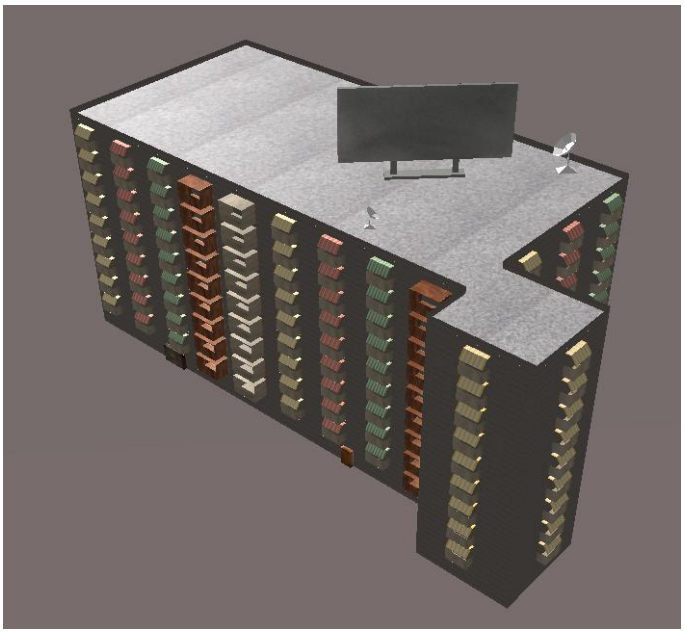

Figura 4-13: Modelo con errores al situar hijos en las esquinas

Para calcular dichos centros se ha ideado la siguiente fórmula. Siendo centroInicial 0 para construir un hijo en el inicio de la cara (izquierda) y 1 para hacerlo en la parte final (derecha), calculamos el centroFinal del hijo como:

CentroFinal = max (anchoHijo * 0.5 / anchoPadre, min (centroInicial , 1 - anchoHijo * 0.5 / anchoPadre))

Siguiendo la fórmula cuando el valor inicial sea 0 el centro pasará a estar en la posición proporcional de la mitad del ancho del hijo respecto al padre, dicho valor estaría cercano al que tomaría el primer hijo de la aparición en fila (0.25) mientras que para el hijo a la derecha el valor sería 1 menos esa proporción estando cercano al del segundo hijo de la aparición en fila (0.75).

Para cada cara del polígono se generarán hijos siempre que se no se cumplan las condiciones ideadas que se enumeran a continuación.

Se ha definido que la base no tenga hijos a través de un parámetro pasado al constructor.

El tamaño de la cara del polígono que estamos creando tiene la anchura mínima permitida para los hijos por lo que sabemos que es el polígono más pequeño que podemos crear y por tanto no puede tener hijos.

Un valor aleatorio que definimos con la fórmula valorAleatorio > (0.5 - iteración * 0.1), de esta forma cuando el valor aleatorio sea menor se creará un hijo y cada vez que se profundice más en el árbol de hijos habrá menos probabilidades de que se generen de forma que en la quinta iteración no permitiremos que se creen para que el edificio no parezca irreal.

Una vez construida la base se irían construyendo las plantas añadiendo vértices en las mismas posiciones de la base, pero a una altura mayor, dicha altura vendría indicada según la altura total del edificio y la altura de cada planta. La estructura de las plantas está

guardada en un conjunto de vértices lo cual nos hace muy sencillo clonarla para el siguiente piso o eliminar hijos para crear terrazas o azoteas intermedias.

Hay que indicar que, para generar azoteas intermedias, es decir, a menor altura que el techo, se ha planteado una poda aleatoria cuya probabilidad de ocurrir es indicada por parámetros y que puede ser nula. A continuación, se muestra un ejemplo de la versión final donde se nos presenta un modelo de edificio con una base compuesta de varios polígonos y que contiene azoteas intermedias.

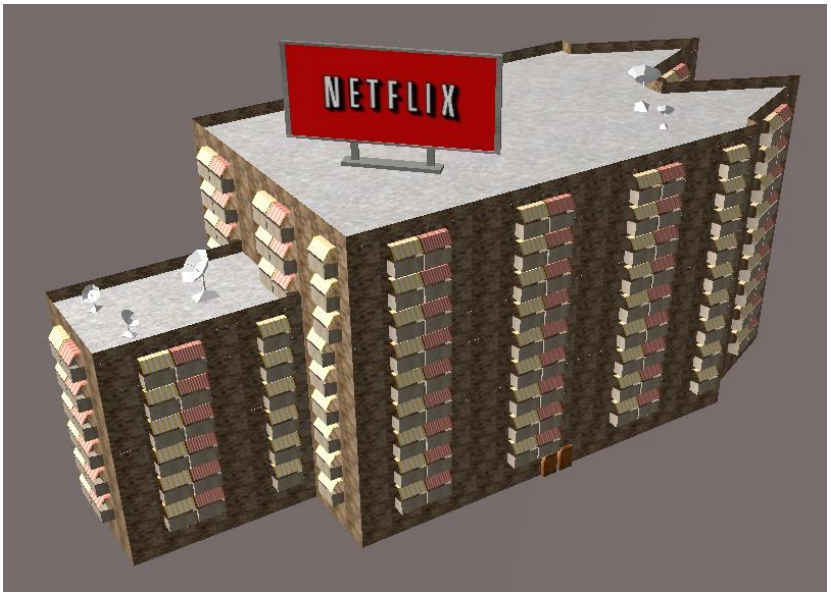

Figura 4-14: Modelo final para un edificio con base compleja y azoteas intermedias

Como ya se ha mencionado se ha utilizado ColladaLoader para cargar los modelos de elementos decorativos en forma de mesh o mallas de vértices que se situarán de forma aleatorio pero coherente sobre la estructura creada del edificio.

En primer lugar, hablaré sobre los elementos decorativos de la fachada que son básicamente ventanas bien en forma de balcón o de terraza. Calculamos que el ángulo sea el mismo de la cara donde se van a situar y las orientamos hacia fuera, tras esto nos faltará hallar la posición de cada ventana, puesto que pueden usarse varios modelos de ventana de diferente tamaño tendremos que tenerlo en cuenta a la hora de hallar la posición de cada una. Para hallar la cantidad de ventanas que podremos situar en la cara dividimos el ancho de dicha cara por la suma de la separación de las ventanas, valor que nos vendrá dado por parámetros, y el tamaño de la ventana más grande.

Ahora que conocemos cuantas ventanas podremos situar tenemos que hallar la posición que ocupará cada ventana, para esto seguimos la siguiente fórmula donde el numero de la ventana comienza desde 0 y que nos dará la proporción de la cara donde situar el centro:

Centro = (numVentana + 0.5 / numVentanasADibujar)

De esta forma si solo hubiera una ventana en la cara se situaría exactamente en el centro. Además, para la opción de situar las ventanas en pares una vez obtenido este centro dibujaríamos la primera ventana con su último vértice sobre el centro y la segunda con el primer vértice sobre el centro.

Por último, hay que señalar que se pueden dar casos en los que, aunque la agrupación sea par, el número de ventanas que podamos dibujar sea impar; en este caso se ha decidido que la última ventana más cercana al primer vértice de la cara donde se añaden las ventanas sea de agrupación simple, en la figura 4-15 puede verse un ejemplo de esto.

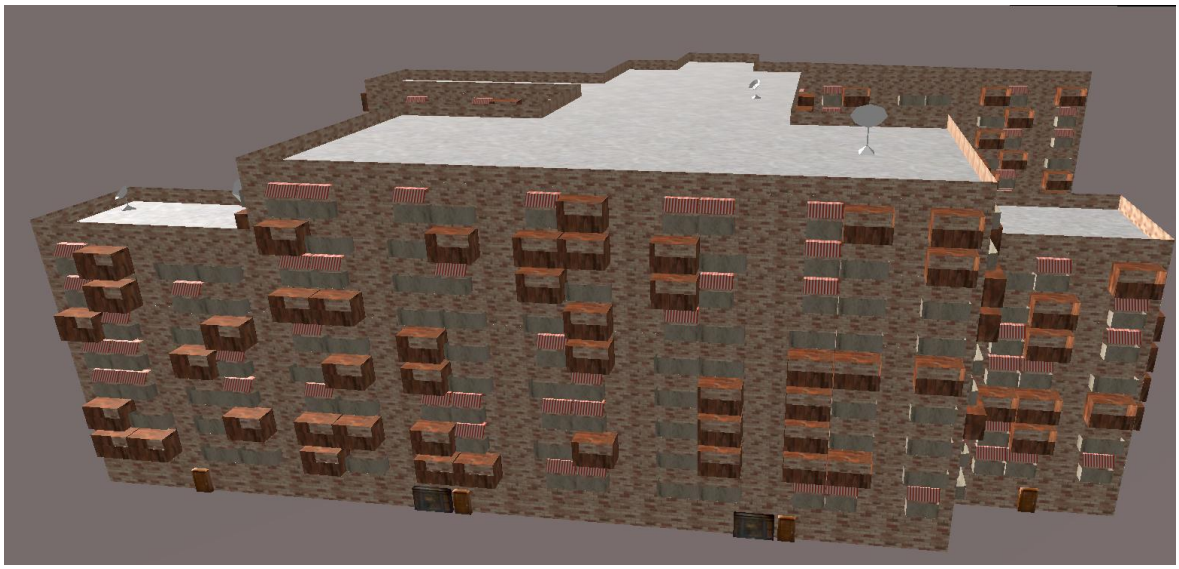

Figura 4-15: Modelo con agrupación de ventanas par y una columna simple

También se puede ver que las puertas al ser tratadas como ventanas de primera planta también se agrupan en pares y que la separación al ser modificada para ser mayor y aumentar el realismo hace que en los hijos solo aparezca una puerta en lugar de dos.
Ahora comentaré los elementos decorativos que se sitúan en las azoteas, en esta versión inicial la librería ofrece modelos para dos tipos de antena, grande y pequeñas, y una
Valla publicitaria con un anuncio de la plataforma Netflix, como ya se ha mencionado con anterioridad se pueden añadir más modelos por parte del usuario para sus edificios personalizados.

Para obtener la posición de un objeto crearemos valores aleatorios dentro de la Bounding Box de techo sobre el que se situarán, la rotación del objeto también será aleatoria de forma que por ejemplo si hay múltiples antenas no apuntarán todas hacia la misma dirección. A partir de esa posición aleatoria obtendremos 4 vértices que encierren el modelo del elemento a añadir, comprobaremos que dichos cuatro vértices no se encuentren fuera de la Bounding Box y en caso de ser válidos se añadirán a la geometría.

Hay que tener en cuenta que una vez añadido un elemento su zona será no válida para los siguientes elementos de forma que no se superpongan entre ellos. Además, al ser valores aleatorios es posible que encontrar una posición válida para un elemento lleve un alto tiempo o que en el caso de azoteas intermedias muy pequeñas sea imposible situar dicho elemento por ello se ha añadido un número máximo de iteraciones para que el algoritmo no se encuentre en un bucle infinito intentando situar elementos en un espacio imposible.

En la texturización del modelo del edificio se utilizan tres tipos de texturas: la textura de la fachada, la textura del techo y el mapa de texturas de los elementos decorativos. Para las imágenes de las texturas se ha elegido un tamaño de 256x256 que da un buen resultado visual y da buen rendimiento al ser una potencia de 2.

A la hora de texturizar las paredes hay que tener en cuenta que los edificios tienen patrones que se repiten constantemente por ejemplo los ladrillos de una fachada, es por esto que las paredes se representan repitiendo la textura elegida para la fachada múltiples veces de forma horizontal hasta ocupar el tamaño del modelo.

La textura elegida tendrá una altura equivalente al parámetro de altura de cada planta, al ser texturas cuadradas (256x256) el ancho será el mismo. Como se ha mencionado anteriormente las texturas utilizadas son idealmente repetibles o "tileables" lo que permite que la textura repetida no produzca un efecto visual de irregularidad en la textura.

A pesar de esto es bastante probable que el ancho de una fachada no sea múltiplo de la altura de las plantas, por ejemplo, para una altura de 5 y un ancho de 23 la última textura solo se dibujaría en un 60%. Para que se mantenga la repetición total de la textura se redondea de forma que en el caso anterior se repetiría la textura 5 veces haciendo más estrecha cada repetición, pero sin afectar al aspecto visual.

Al usar está solución hay que controlar el caso en el que la fachada sea estrecha como para no pintar ni una vez la textura, para estos casos se usa solo el porcentaje que cubra dicha pared ya que estrechar tanto la textura si afectaría al aspecto visual.

Entre los parámetros del edificio como ya hemos mencionado está indicar la altura de una planta de forma que podrían darse casos en que las plantas sean muy grandes para pintar una solo vez la textura, en estos casos se realiza algo similar a la solución anterior, tomamos como valor de referencia el valor por defecto de la altura de una planta y si por ejemplo la altura de la planta especificada por parámetro representa 1.6 veces el valor por defecto se pintarán dos filas de textura para la misma planta, sin embargo en este caso no es una cuestión de estrechar sino de reducir el tamaño; es decir en primer lugar decidimos cuantas filas de la textura se usarán y en base a eso se establece el tamaño de la textura, tras ello se estrecha en función del ancho de la cara del edificio.

En esta imagen puede verse como al hacer que la texturización sea con una textura "tileable" permite que las esquinas encajen perfectamente la generar el modelo.

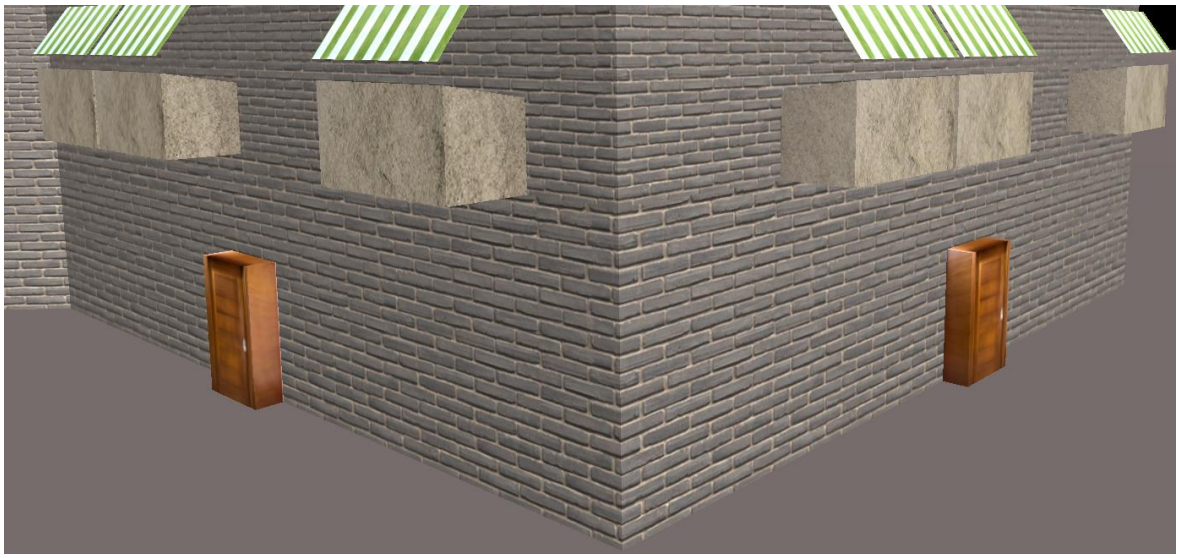

Figura 4-16: Texturización de la fachada haciendo que encaje en las esquinas

Anteriormente se ha explicado que la geometría del techo está construida a través de triángulos que juntan todos los vértices de la figura del tejado hasta cubrir todo el espacio de la última planta. Tomemos el ejemplo de un techo con geometría rectangular de 10 unidades de ancho y 20 de alto, si el tamaño de nuestra textura es de 2x2 unidades necesitaremos 5 repeticiones horizontales y 10 verticales, el techo está constituido por dos triángulos de forma que sus vértices son T1 = ({0,0} {20,0} {20,10}) T2 = ({0,0} {0,10} {20,10}). Tendremos que sustituir el valor máximo de cada eje por el número de repeticiones en vertical y horizontal de forma que los valores cambiarían a T1 = ({0,0} {10,0} {10,5}) T2 = ({0,0} {0,5} {10,5}), el resto de valores se ponderan entre el valor mínimo y el máximo en este caso los ceros no varían. En otras formas se haría el proceso de la misma forma aplicándolo sobre los triángulos en sentido horario, para la librería el tamaño de la textura es de 20x20 unidades que da un buen rendimiento visual como se puede ver en la figura 4-17.

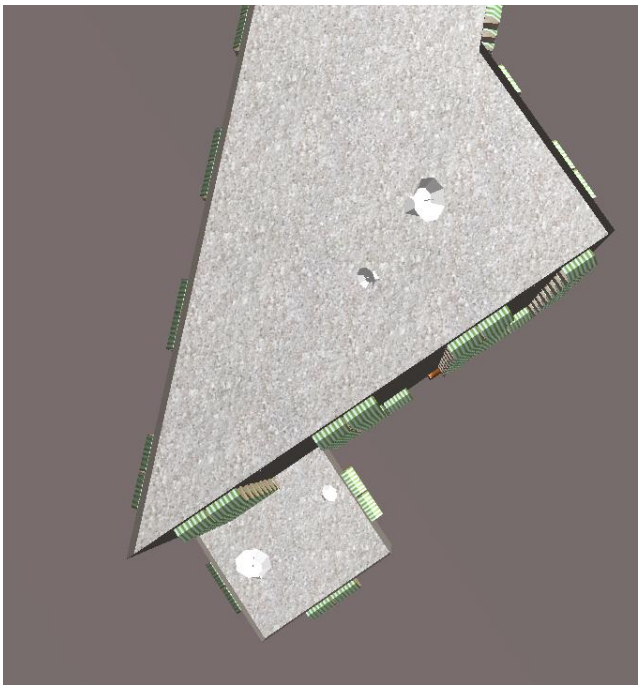

Figura 4-17: Ejemplo de texturización del techo de un edificio

En el caso de las texturas para los elementos decorativos no es necesario que sean "tileables" ya que están mapeadas con el modelo para dar un aspecto creíble, para reducir el trabajo de la CPU manejando imágenes se añaden las texturas de los elementos decorativos en un elemento <canvas> que se podrá consultar a lo largo del algoritmo para la texturización. En el elemento se guardarán un máximo de 16 texturas es decir se pueden añadir hasta 16 elementos decorativos diferentes los cuál se ha estimado que es suficiente para este proyecto. En el fichero Texturas.js está la funcionalidad que permite consultar el <canvas> para obtener una textura en concreto que aplicar sobre un elemento decorativo. En nuestro simulador de prueba se ha hecho visible este elemento <canvas> para mostrar su funcionamiento, aunque no es necesario que el usuario haga lo mismo.

Figura 4-18: Imagen del simulador donde se hace visible el elemento canvas con las texturas de los elementos decorativos.

4.3.2 Algoritmos matemáticos

Entre los algoritmos matemáticos implementados el más destacable es aquel que nos permite saber que un punto se encuentra dentro de un polígono lo cual se utiliza varias veces a lo largo del algoritmo principal, para ello he usado el algoritmo de Ray Casting.

Para este algoritmo trazamos un segmento entre el punto a comprobar y un punto que se encuentra fuera del polígono, para ello simplemente obtenemos el valor de la X y la Y del polígono y sumamos 1. Ahora comprobamos cuántas veces se cruza el segmento trazado con las aristas del polígono. Para el caso de un edificio sería con todas las aristas de la base y los hijos.

Una vez hechos todos los cruces con las aristas del polígono contamos las veces que se han cruzado de forma que si la cantidad es impar el punto está en el polígono y si es par está fuera, en caso de encontrarse sobre un borde del polígono lo consideramos como estar dentro. Para aclarar esta explicación añado una imagen para un punto de ejemplo que se encuentra fuera del polígono y otro que se encuentra fuera.

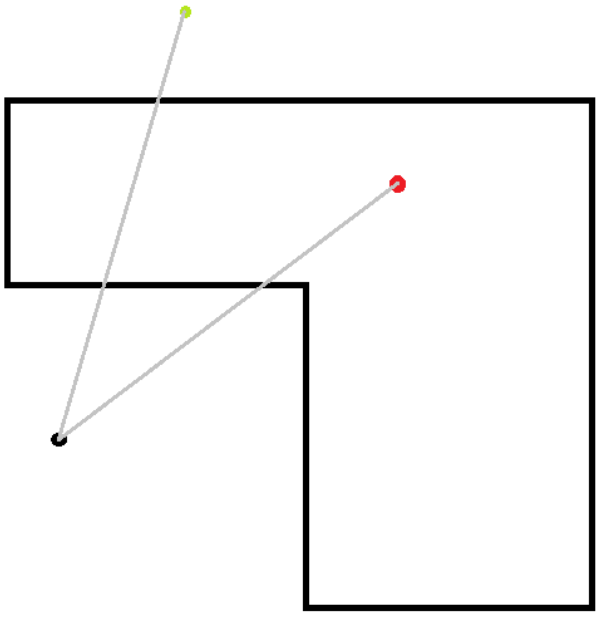

Figura 4-19: Figura ejemplificando el Ray Casting

En este ejemplo como se puede ver el segmento del punto exterior con el punto verde cruza dos veces por lo que el verde está fuera y el del rojo que cruza una vez está dentro.

A continuación, comentaré de una forma más breve los diferentes algoritmos matemáticos que he necesitado implementar.

Para ponderar un porcentaje entre dos valores he implementado una función tomamos como valor mínimo el más bajo, después multiplico el porcentaje por la diferencia del valor mayor por el menor y se lo sumo al menor de esta forma obtendremos el valor que se encuentra entre el menor y el mayor en un porcentaje específico.

He necesitado una simple comprobación para revisar que un valor se encuentre entre otros dos, en este caso he añadido una precisión personalizada según el caso para aceptar que un valor esté entre dos con una precisión de un solo decimal, pero para tres decimales no, por ejemplo, sería el caso del valor 15,123 en el intervalo de valores 14 y 15,1.

También he implementado una solución de Cramer para sistemas de ecuaciones de dos incógnitas siguiendo la fórmula original de forma que para las ecuaciones
ax + by = c
dx + ey = f
El valor de las incógnitas sería x = (c*e – b*f) / (a*e-b*d) y= (a*f – c*d) / (a*e-b*d)

Así mismo he necesitado hallar la recta que pasa por dos puntos en un plano, para ello de nuevo he seguido la fórmula matemática adaptándola al código. La pendiente de dicha recta se calcularía como (punto2.y – punto1.y) / (punto2.x – punto1.x);

Por último, he implementado una función para comprobar que dos polígonos se intersecan, para ello itero sobre todos los vértices del primer polígono y creo rectas con los inmediatamente siguientes con el uso de la función anteriormente comentada, cada recta del primer polígono se compara con todas las rectas del segundo polígono calculadas de la misma forma, para compararlas usamos la función de Cramer con las dos ecuaciones de cada pareja de rectas. Una vez tenemos el resultado de Cramer comprobamos que se encuentre entre las dos parejas de vértices de cada polígono, si esto ocurriera los polígonos estarían intersecándose.

4.3.3 Diagrama de módulos

En este apartado adjunto un diagrama de los diferentes módulos que participan en la funcionalidad de la librería, he excluido los módulos referidos al simulador ya que no es el centro del proyecto.

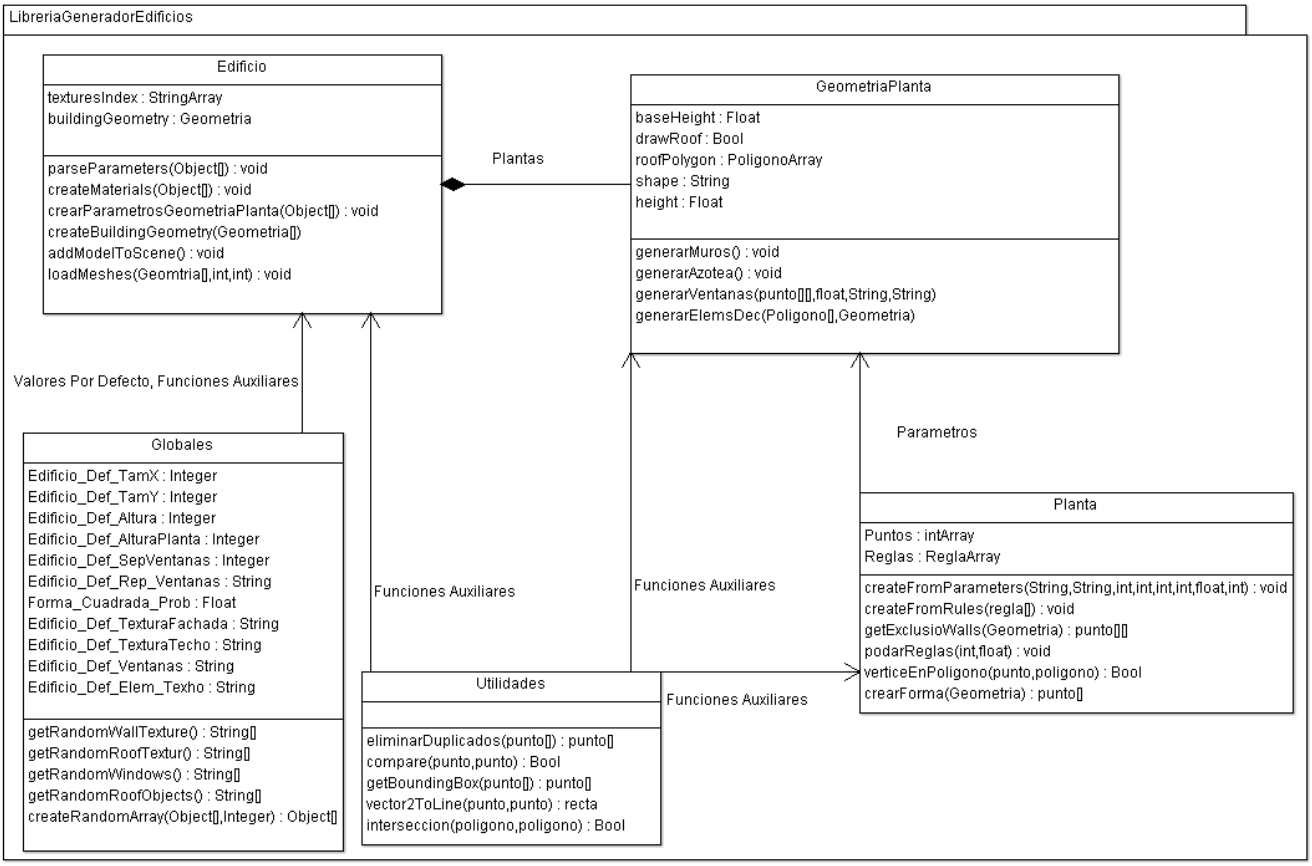

Figura 4-20: Diagrama de módulos de la librería

5 Desarrollo

Como ya se ha mencionado con anterioridad se ha implementado la librería de forma modular para facilitar el desarrollo y ampliación de funcionalidad, a continuación, se explicará paso a paso la implementación del algoritmo comentando los aspectos más importantes y como encajan en el diseño del proyecto, incluyendo la parte de simulación.

Dada la gran extensión del código implementado he decido incluir como anexo el pseudocódigo de los ficheros que contienen la funcionalidad del generador y añadir en dicha sección un enlace al código que se encontrará almacenado en una plataforma externa.

En primer lugar, se mencionarán los ficheros utilizados para la creación del simulador. Como se puede suponer se han utilizado un fichero html index y un fichero de estilos css, sobre ellos solo cabe destacar que se han indicado los diferentes parámetros para el modo paramétrico en una sección que se expande cuando se selecciona dicho modo, también se hace visible, como se comentó en la texturización de elementos decorativos, el elemento <canvas> del que se obtienen las texturas a aplicar sobre cada elemento.

El fichero controles.js implementa la funcionalidad para controlar la cámara que nos permite ver la escena y con el uso de Jquery lee los parámetros elegidos y llama a la librería para generar el modelo del edificio y renderizarlo sobre la escena. Dentro de los controles del simulador también tenemos el fichero cámara.js que sería el encargado directo de modificar la posición u orientación de la cámara para poder visualizar el modelo de edificio generado desde todas las posiciones deseadas.

Como ya se mencionó anteriormente el fichero texturas.js se encarga de introducir las texturas de elementos decorativos actuales en el elemento canvas y de consultarlas cuando sea necesario aplicarlas usando el mapeo del modelo sobre su respectivo elemento, también se encarga de eliminar y añadir las texturas cada vez que se genera un modelo nuevo.

Por último, el fichero principal main.js es el que se encarga de crear la escena, situar un plano que haga de terreno, iluminar la escena y llamar por primera vez a la librería para que aparezca un edificio por defecto al iniciar el simulador.

En la figura 5-1 se puede ver una imagen del simulador con las opciones mencionadas.

Figura 5-1: Imagen del simulador implementado

Una vez explicada la funcionalidad del simulador pasaré a comentar los diferentes ficheros que componen la funcionalidad de la librería.

En primer lugar, hay que mencionar dos ficheros auxiliares; utilidades.js que contiene funciones de uso común en el algoritmo o implementaciones de funciones matemáticas que reducen la cantidad de código en los ficheros principales. A continuación, se comentarán las tres funciones más importantes de este fichero.

La función "eliminarDuplicados" realiza la función ya comentada de eliminar los vértices del modelo que son duplicados, esto ocurre como ya se ha explicado al situar un hijo justo en la esquina, izquierda o derecha, de la cara del padre de forma que uno de los vértices del hijo se situará en el mismo lugar que uno del padre.

La función "getBoundingBox" nos devuelve los límites que tenemos para construir el modelo del edificio lo cual nos permite calcular la texturización de la fachada y controlar que los elementos decorativos del tejado no se salgan de los límites del edificio.

La función "intersección" recibe los arrays de vértices de dos polígonos y comprueba si se encuentran en intersección, para esto se construyen segmentos con los vértices contiguos de ambos polígonos y se comprueba la intersección con los segmentos del otro polígono. Esta función es utilizada para comprobar que los hijos de distintos padres no se choquen entre ellos.

El fichero globales.js contiene valores por defecto para el edificio además de funciones que nos devuelven conjuntos aleatorios de las texturas y los elementos aleatorios por defecto mediante arrays de tamaño variable, esto permite que en la generación random se usen distintos tipos de texturas para la fachada y techo y cantidad variable de elementos decorativos.

El fichero edificio.js es el fichero principal de la librería y es la clase que se utiliza para generar el modelo de un edificio pasando por parámetros todos los valores que se quieran especificar. En primer lugar, se utiliza una función para parsear parámetros de manera que

se recogen los parámetros especificados y se establecen como aleatorios o por defecto, según el caso, el resto de parámetros para la construcción del modelo.

Tras esto se cargan las texturas de la fachada, el techo y los distintos elementos decorativos y se establece una relación entre cada textura y la geometría del elemento sobre el cuál se mapea la textura para dar un aspecto visual creíble, de esta forma se generan los materiales que posteriormente se usarán para la texturización.

Una vez hecho esto se pasa a la función principal que construye la geometría del edificio, se crea la geometría de cada planta con el fichero plata.js que se explicará posteriormente, en esta función se indica si la siguiente planta sufrirá una poda y por tanto se creará una azotea intermedia, de la misma forma se comprueba si es necesario generar un techo bien intermedio o final en esta planta, además se añadirán las ventanas a la fachada de la forma que se explicó en el apartado de diseño y se añadirán los elementos decorativos del techo en caso de que haya, de nuevo, un tejado intermedio o sea el final del edificio. Por último, se dibuja la pared de tejado final cuya texturización sigue el mismo proceso que la fachada salvo que en este caso se hace por ambas caras del material ya que ambas son visibles para el usuario.

Una vez se ha construido toda la geometría y se le han aplicado texturas el edificio se crea una mesh con la funcionalidad de Three.js cuyos parámetros serán la geometría construida para el edificio y los materiales generados al cargar las texturas, posteriormente dicha mesh se añade a la escena para visualizar el modelo en el simulador.

Ahora pasaré a comentar el fichero geometriaPlanta.js que recibe unos parámetros esenciales desde el fichero anterior para crear la geometría correctamente, hay que indicar que dichos parámetros esenciales se obtienen a través del fichero planta.js que está destinado a traducir la forma deseada para la planta a un conjunto de vértices que permitirán generar la geometría de cada planta, los vértices nos indican la forma actual del edificio, se itera sobre las diferentes caras de hijos y padres y se comprueba por los métodos explicados en el diseño si se va a añadir un nuevo hijo, en caso afirmativo es este fichero el que se encarga de revisar si el tamaño del hijo es adecuado o si el padre puede tenerlo, también es el encargado de podar los hijos de forma que eliminar los vértices correspondientes a dicho hijo para que no se aparezca en la geometría y se dibuje un techo para él, hay que aclarar que nunca se poda un hijo que no sea final, aunque de nuevo esto se puede ver en algunas construcciones actuales he decidido no hacerlo para mantener el realismo en los edificio más comunes.

Una vez se tienen los vértices que formarán esta planta se llama a geometríaPlanta.js que en primer lugar construye la fachada de la planta a través de triángulos hasta cubrir el ancho de dicha planta, una vez hecho tendremos el conjunto de vértices en forma de triángulos que forman las cuatro caras de la planta. Posteriormente se dibuja el tejado con el método explicado en detalle en el diseño de forma que los vértices de la forma final se van uniendo formando triángulos hasta cubrir la totalidad de la superficie del techo y facilitando la texturización al usar triángulos en este proyecto. Además, hay que indicar que este fichero es el que se encarga de insertar las ventanas y los elementos decorativos del edificio mediante funciones que se llaman desde edificio.js una vez ya se tiene la geometría de la planta en cuestión.

6 Integración, pruebas y resultados

Como ya se ha indicado anteriormente se ha utilizado la librería Three.js además de un proyecto asociado ColladaLoader.js para realizar tareas como crear triángulos a partir de tres vértices, crear materiales para los elementos del edificio o cargar las texturas a utilizar, por lo tanto es necesario que se incluyan ambas librerías para utilizar la librería implementada, fuera de ello solo habría que incluir la librería desarrollada por el método habitual para una librería javascript, idealmente en el momento de usarse en proyectos externos la librería pasaría a ser un solo fichero ya que el proyecto estaría finalizado. Una vez hecho esto solo habría que solicitar el modelo de un edificio indicando la escena donde quiere añadirse y aspectos como la rotación para que el edificio esté orientado hacia una zona en especial, aparte de ello habría que especificar los parámetros que se quieran para obtener el edificio más detallado posible, en este aspecto pasarán a explicarse de forma práctica los tres tipos de generación que ya se han visto a lo largo de la presente memoria.

En el modo por defecto tan solo tendremos que indicar el parámetro de modo y obtendremos un edificio sencillo que nos puede servir para comprobar que hemos integrado correctamente la librería, también es el edificio que se obtiene en caso de no especificar parámetro alguno.

```
newBuilding = new Edificio({'mode': "default"});
```

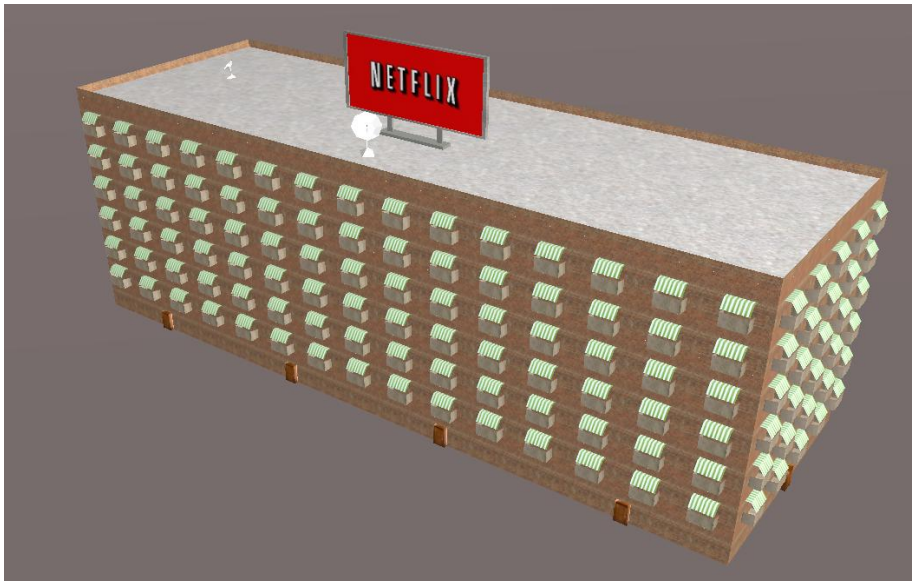

Figura 6-1: Imagen del modelo de edificio por defecto

En el modo por defecto de nuevo solo habría que indicar que queremos ese modo ya que todos los parámetros los generará la librería.

```
newBuilding = new Edificio({'mode': "random"});
```

Por último, el modo más útil y que es el principal del proyecto en el cuál podemos indicar la cantidad de parámetros que queramos, a continuación, mostraré un par de ejemplos de edificio creado por parámetros y el resultado obtenido.

```
newBuilding = new Edificio({'mode': "parameters",
'height': 50,
'sizeX': 120,
'sizeZ': 60,
'floorHeight': 50,
'windowsGroup': “pairs”,
'windowsSeparation': 5,
'probAzoteasIntermedias': 0.25
'minChildTam': 25,
'maxChildTam': 100
'windowRepetition': “lineal”,
'textureWall': “/texturas/fachada/pared3.jpg”
'textureRoof': “/texturas/techo/techo1.jpg})
```

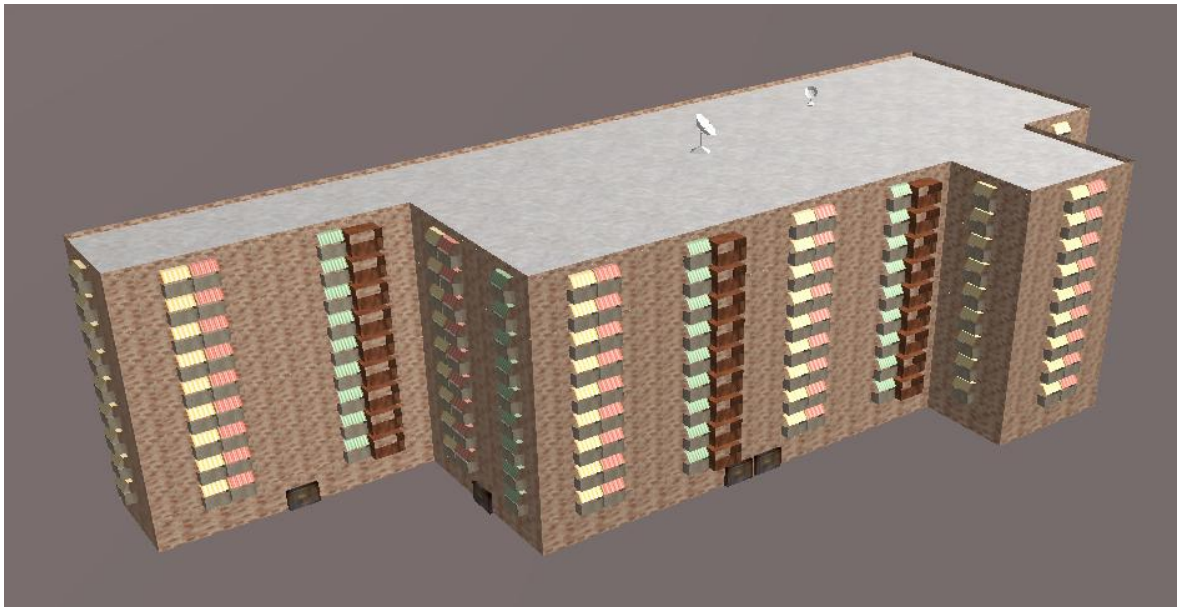

Figura 6-2: Modelo del primer edificio de prueba

Podemos ver como las ventanas se han agrupado en pares y el uso de texturas es en forma lineal repitiéndose por columnas, además al establecer una probabilidad de 0.25 es normal que no hayan aparecido azoteas intermedias.

```
newBuilding = new Edificio({'mode': "parameters",
'height': 80,
'sizeX': 90,
'sizeZ': 40,
'floorHeight': 6,
'windowsGroup': “normal”,
'windowsSeparation': 8,
'probAzoteasIntermedias': 0
'minChildTam': 101,
'maxChildTam': 101,
'windowRepetition': “random”,
'textureWall': “/texturas/fachada/pared4.jpg”
'textureRoof': “/texturas/techo/techo2.jpg})
```

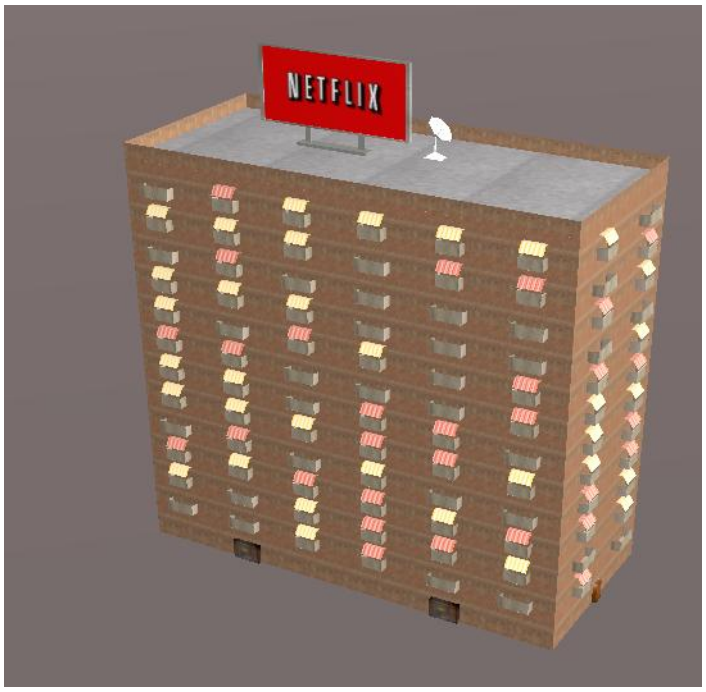

Figura 6-3: Modelo del segundo edificio de prueba

En este caso las ventanas se agrupan de forma simple y las texturas se aplican de forma aleatoria, el tamaño del edificio ha cambiado y al indicar que el tamaño mínimo de los hijos sea del ancho del edificio+1 no se genera ninguno.

En cuanto al rendimiento se han realizado varias pruebas con distintos tamaños de edificios como se puede comprobar el tiempo para la creación es bastante aceptable y no sobrepasa los para el tamaño más grande probado

Número de vértices	Tiempo de generación (ms)
3179	302
4698	515
9213	427
12038	608
28226	978
50826	1591

Tabla 3-3: Rendimiento de generación de edificios

Por último, añadimos la imagen del edificio de mayor tamaño de la tabla anterior para poder comprobar la capacidad de creación de la librería.

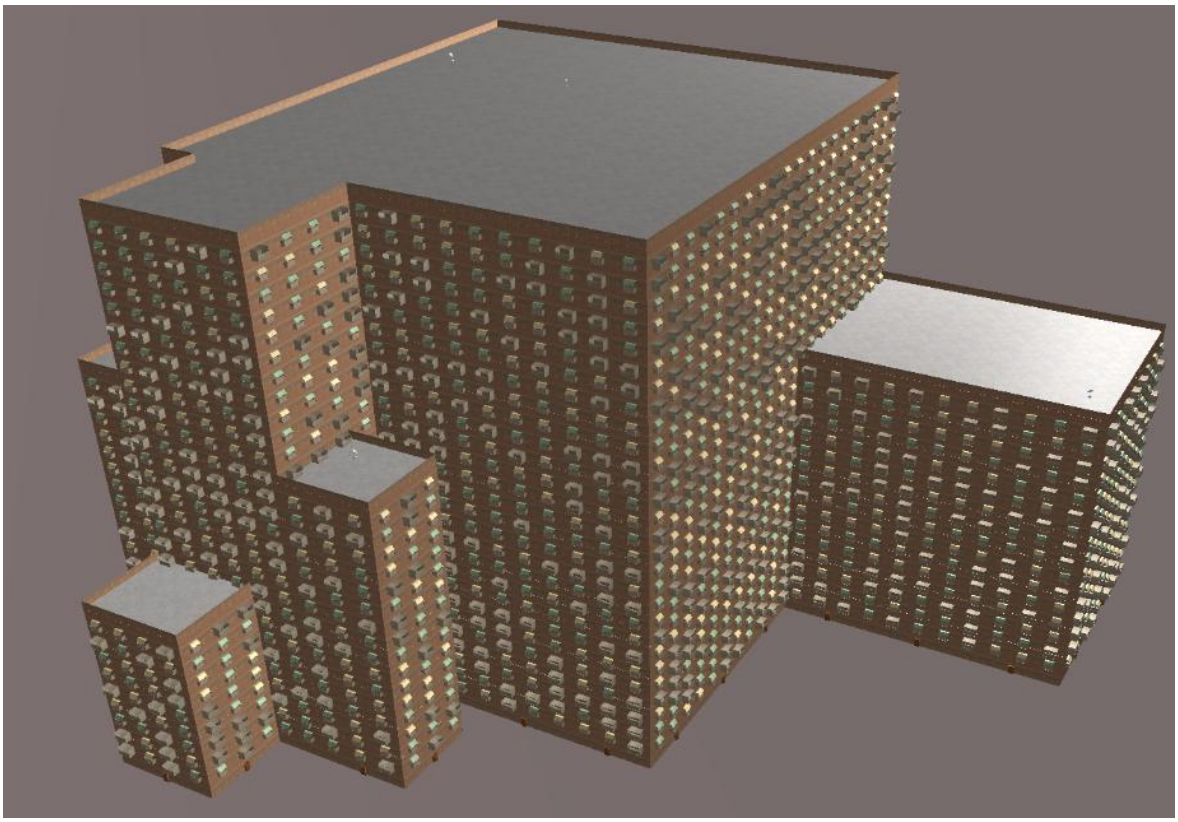

Figura 6-4: Modelo del tercer edificio de prueba

7 Conclusiones y trabajo futuro

7.1 Conclusiones

En primer lugar, creo que se ha implementado con éxito el proyecto planteado y que se ha hecho un uso eficiente de las funcionalidades ofrecidas por la librería de código abierto Three.js, por otra parte, considero que el aspecto visual de los modelos de edificio es bastante realista y creo que el resultado es satisfactorio con lo ideado inicialmente.

El uso de la librería por parte de un usuario externo es intuitivo y permite que el usuario pueda detallar hasta el punto que quiera las características del edificio, además puede incluir modelos que haya diseñado el usuario para que sus edificios sean aún más personalizables.

El facto aleatorio se ha incluido en cantidad, pero de forma controlada lo cual permite que se genere una gran cantidad de edificios de características variables, pero manteniendo el realismo del modelo para que el usuario pueda obtener diferentes edificios que encajen con sus necesidades.

Por último, como hemos visto en la sección de rendimiento el tiempo para generar los edificios es bastante reducido, lo normal es que el usuario quiera un edificio de tamaño medio de alrededor de 10000 vértices lo cual se genera en unos 500 milisegundos mientras que si quisiera un edificio extremadamente grande de 50000 vértices tarda en generarse 1,5 segundos tiempo que es aceptable para modelos que no suelen solicitarse.

7.2 Trabajo futuro

En esta última sección mencionaré algunas mejoras que se podrían realizar sobre la librería para ampliar la funcionalidad implementada en este proyecto.

Añadir limitaciones para que el edificio solo genere hijos en algunas direcciones para poder adaptarse al entorno que esté utilizando el usuario.

Implementar la construcción de tejados de tejas en lugar de azoteas, esto requiere una gran complicación por lo que no se ha planteado en este proyecto, en primer lugar, elementos como las antenas deberían situarse en una orientación específica para no chocar con el tejado, además la misma construcción del tejado implicaría levantar superficies en forma de triángulos que se unieran y formarán el dicho tejado.

Añadir elementos decorativos en la primera planta como jardines o verjas, esto sería similar a lo hecho para los elementos del tejado, pero en este caso habría que tomar una parte del límite para el edificio para dichos elementos de la primera planta lo que haría que el edificio tuviera menor tamaño. También podrían hacerse elementos de azoteas que dependan de su forma por ejemplo piscinas de mayor o menor tamaño.

Texturas y elementos decorativos generados de forma procedimental, en el diseño actual las texturas se pasan por parámetro y los elementos se reciben en forma de textura y modelo, en un diseño más avanzado podrían generarse texturas procedimentales a partir de varias texturas simples, y los elementos podrían pasarse como una base donde un balcón este dividido en la estructura, el toldo, objetos dentro del balcón que puedan situarse de

una forma u otra para que parezca que cada balcón es único pero todos partiendo de una base.

Comprobar que los parámetros que introduce el usuario no son demasiado costosos para optimizar la librería de forma que se pueda avisar al usuario que a causa de parámetros como por ejemplos introducir sus propios elementos decorativos o ventanas que sean excesivamente complicadas es decir que se compongan de un gran número de vértices puede causar un alto retraso en la generación del modelo esperado.

Referencias

[1] **Chris Baker, "'No Man's Sky': How Games Are Building Themselves", visitado el 1 de abril de 2017**
http://www.rollingstone.com/culture/news/no-mans-sky-how-games-are-building-themselves-w433492
[2] **Jos Dirksen, "Learning Three.js – the JavaScript 3D Library for WebGL Second Edition", Marzo 2015.**
[3] **"Fractal", visitado el 1 de abril de 2017**
https://es.wikipedia.org/wiki/Fractal
[4] **"Alfombra de Sierpinski", visitado el 1 de abril de 2017**
https://es.wikipedia.org/wiki/Alfombra_de_Sierpinski
[5] **"Sistema-L", visitado el 1 de abril de 2017**
https://es.wikipedia.org/wiki/Sistema-L
[6] **Yoav Parish y Pascal Müller, "Procedural Modeling of Cities", visitado el 2 de abril de 2017**
https://graphics.ethz.ch/Downloads/Publications/Papers/2001/p_Par01.pdf
[7] **Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer y Luc van Gool, "Procedural Modeling of Buildings", visitado el 2 de abril de 2017**
http://peterwonka.net/Publications/pdfs/2006.SG.Mueller.ProceduralModelingOfBuildings.final.pdf
[8] **Jess Martin, "Procedural House Generation: A method for dynamically generating floor plans", visitado el 2 de abril de 2017**
http://axon.cs.byu.edu/Dan/673/papers/martin.pdf
[9] **"Three.js – documentation", visitado el 3 de abril de 2017**
https://threejs.org/docs/
[10] **"Blender", visitado el 3 de abril de 2017**
https://www.blender.org/
[11] **"Voxels", visitado el 3 de abril de 2017**
https://es.wikipedia.org/wiki/V%C3%B3xel
[12] **"Malla poligonal", visitado el 3 de abril de 2017**
https://es.wikipedia.org/wiki/Malla_poligonal
[13] **Jess Martin, "Algorithmic Beauty of Buildings Methods for Procedural Building Generation", visitado el 20 de mayo de 2017**
http://digitalcommons.trinity.edu/cgi/viewcontent.cgi?article=1003&context=compsci_honors
[14] **George Kelly Hugh McCabe, "Citygen: An Interactive System for Procedural City Generation", visitado el 20 de mayo de 2017**
http://www.citygen.net/files/citygen_gdtw07.pdf
[15] **George Kelly Hugh McCabe, "A Survey of Procedural Techniques for City Generation", visitado el 20 de mayo de 2017**
https://pdfs.semanticscholar.org/1e9c/1349b91d9352148a4567d172df7a965c180c.pdf
[16] **Kenichi Sugihara y Zhenjiang Shen, "Automatic Generation of 3D Building Models with Efficient Solar Photovoltaic Generation", visitado el 22 de mayo de 2017**
https://www.jstage.jst.go.jp/article/irspsd/5/1/5_4/_pdf

Glosario

GPU	Graphics Processor Unit.
Procedimental	Generado a través de un algoritmo.
Parsear	Proceso por el cual se analizan uno o varios valores de entrada para obtener de ellos información útil.
Mapear	En concreto mapear una textura sobre un modelo 3D, es la técnica de relacionar los vértices y superficies del modelo para que una textura se aplique sobre él adaptándose a los diferentes tamaños que pueda tomar el modelo.
Renderizar	Proceso para generar una imagen a partir de un modelo 3D, en nuestro caso aplicable a generar las texturas del edificio.
Poda Aleatoria	Proceso por el cuál según un factor aleatorio se elimina alguno de los polígonos extremos de la malla de polígonos que constituyen la forma del edificio.
Mesh	Malla de polígonos que constituyen una estructura dentro del edificio, puede referirse a un elemento decorativo, a la forma del edificio o al edificio completo.
Bounding Box	Caja imaginaria que limita la superficie o volumen que puede ocupar un elemento dentro de un espacio, en nuestro caso lo limita según unos parámetros de altura, anchura y longitud.
Tileable	Se dice de la textura que por su estructura permite repetirse de forma horizontal y vertical sin dar la sensación de que el resultado final esté constituido por una textura de menor tamaño.
Texturización	Proceso por el cual se aplica una imagen, que representa una textura, sobre un modelo 3D.

Anexos

A Manual de uso

Para hacer uso de esta librería en primer lugar deberá incluir los ficheros js que constituyen los módulos de la librería de generación procedimental de edificios, en una versión futura estos módulos se resumirán en un uno para hacer más sencillo su inclusión en el proyecto del usuario. Los módulos a incluir son: Edificio.js, Planta.js, GeometriaPlanta.js, Globales.js y Utilidades.js. También deberá incluir en su proyecto los modelos geométricos y de texturas que la librería ofrece si desea utilizarlos.

Además, deberá añadir ficheros de librerías de código abierto externos a esta librería, dichas librerías son: three.js y ColladaLoader.js

Para hacer uso de la librería deberá crear una escena si no la tenía ya en su proyecto, posteriormente deberá llamar al módulo Edificio, si llama a su constructor sin parámetros recibirá el edificio por defecto de la librería para comprobar si se ha añadido correctamente a su proyecto.

Una vez haya comprobado que el edificio por defecto se añade correctamente podrá crear edificio más personalizados, si indica el modo random recibirá un modelo de edificio aleatorio, con esta opción puede comprobar diferentes modelos de edificios que podrían servirle en su proyecto.

Si desea crear un edificio personalizado deberá indicar los parámetros que desea que sean fijos, contra más especifique más se ajustará el modelo a lo que espere.

Los parámetros que puede especificar en el edificio son los siguientes:

Mode, deberá indicar el valor "parameters" para poder especificar el resto de variables.
Height: Altura total del edificio.
SizeX: Longitud del edificio.
SizeZ Ancho del edificio.
FloorHeight altura de las plantas del edificio.
WindowsGroup, agrupación de las ventanas del edificio puede ser de forma singular; “normal” o en parejas; “pairs”.
WindowsSeparation es la separación entre las ventanas de una planta en caso de agruparse en pares será la distancia de un par a otro.
ProbAzoteasIntermedias es la probabilidad de que la forma base del edificio cambie a una altura determinada y se añada una azotea en la zona donde una extensión de la base deje de aumentar en altura.
MinChildTam, es el tamaño mínimo que debe ocupar una extensión de la base para poder añadirse a ella.
MaxChildTam es el tamaño mínimo que debe ocupar una extensión de la base para poder añadirse a ella
WindowRepetition es el modo en que se repiten las texturas de las ventanas puede ser de forma regular; “lineal” o aleatoria, “random”.

TextureWall es la dirección en la que se encuentra la imagen que se usará para texturizar la fachada del edificio.
TextureRoof' es la dirección en la que se encuentra la imagen que se usará para texturizar las azoteas.

Para introducir elementos decorativos propios ya sean ventanas o elementos del tejado se deberá crear un objeto dentro del objeto que se pasa como argumento al constructor de la librería que contiene todos los demás parámetros de la siguiente manera:

```
newBuilding = new Edificio ({'mode': "parameters",
'height': 80,
'sizeX': 90,
'sizeZ': 40,
'floorHeight': 6,
‘roof’: {
        ‘wallHeightProportion’:0.5,
        ‘objects’: [{
                'mesh': './modelos/elementos/AntenaPequeña.dae?2',
                'texture': './texturas/elementos/antena.jpg'}, {
                'mesh': './modelos/elementos/AntenaGrande.dae',
                'texture': './texturas/elementos/antena.jpg'}]
},
‘windows’: [{
        'mesh': './modelos/balcones/Balcon-Toldo.dae',
        'texture': './texturas/balcones/Balcon-ToldoAmarillo.jpg'}, {
        'mesh': './modelos/balcones/Balcon-Toldo.dae?2',
        'texture': './texturas/balcones/Balcon-ToldoRojo.jpg'}
]})
```

En lo referente al simulador, tenga en cuenta que hay navegadores que tienen políticas contra el uso de archivos locales por lo que si desea probar el simular le aconsejo que utilice un navegador que permita el uso de archivos locales como Mozilla Firefox ya que uno que los bloquee puede causar que aparezcan modelos sin texturizar correctamente.

Los controles de la cámara del navegador son los siguientes:

A: Rotación a la izquierda de la cámara sobre el modelo generado.
D: Rotación a la derecha de la cámara sobre el modelo generado.
W: Zoom hacia el modelo.
S: Zoom alejándose del modelo.
Q: Rotación hacia arriba de la cámara sobre el modelo generado.
E: Rotación hacia abajo de la cámara sobre el modelo generado.
Flecha Izquierda: Movimiento hacia la izquierda independiente del modelo generado.
Flecha Derecha: Movimiento hacia la derecha independiente del modelo generado.
Flecha Arriba: Movimiento hacia delante independiente del modelo generado.
Flecha Abajo: Movimiento hacia atrás independiente del modelo generado.

B Código de la aplicación

Enlace al código de la librería: https://www.dropbox.com/s/2x8dzh4eoxovf0m/TFG-Salas.rar?dl=0

En este anexo incluyo un enlace al código de la librería con el que se puede comprobar la funcionalidad implementada, además presentaré el pseudocódigo de los módulos Edificio, Planta y GeometriaPlanta, he decidido omitir el de los módulos Utilidades y Globales porque no aclaran el flujo del algoritmo que es lo que pretendo con el de los tres primeros.

Módulo Edificio

Parsear parámetros recibidos
Crear materiales en base a los parámetros
Inicializar geometría del edificio
Cargar mallas de puntos de elementos del edificio (ventanas, decoración, etc)
Crear geometría del edificio:

Crear parámetros de la primera planta con el uso del módulo Planta en base a los parámetros recibidos
Repetir

> Variar los parámetros de la planta en función de probabilidades y la altura a la que esté la planta actual
> Crear geometría de la planta con el uso del módulo GeometriaPlanta en base a los parámetros modificados para la planta actual
> Se insertan ventanas con el uso del módulo GeometriaPlanta
> Si hay elementos de decoración
> > Se añaden elementos de decoración con el uso del módulo GeometriaPlanta
> Se fusiona la geometría de la planta con la del edificio
> Si es la última planta del edificio o se ha creado una azotea intermedia
> > Se genera la pared de la azotea
> > Se fusiona la geometría de la pared de la azotea con la del edificio
> Si el parámetro de cambiar la forma de la siguiente planta es true
> > Cambian las reglas de la siguiente planta con el uso del módulo Planta
> Aumenta la altura actual del edificio

Hasta i > número de plantas del edificio

Se añade la geometría del edificio a la escena que el usuario indique por parámetro

Módulo Planta

Construcción por parámetros:

Generamos reglas para la planta en función de los parámetros recibidos
Creamos los puntos que forman la geometría con el uso de las reglas generadas
Calculamos los hijos de la geometría de la planta en base a las reglas

Construcción por reglas:

Clonamos las reglas recibidas
Calculamos los hijos de la geometría de la planta en base a las reglas
Creamos los puntos que forman la geometría con el uso de las reglas recibidas

Obtener muros de azoteas:
Si la planta actual no tiene la misma geometría que la anterior
    Repetir

        Si el hijo actual de la planta anterior no se encuentra en la planta actual añadir a un array de reglas eliminadas

    Hasta i > número de hijos planta anterior

    Repetir

        Añadir los puntos de la regla eliminada actual a un array de puntos para el muro de la azotea

    Hasta i > número de reglas eliminadas

Devolver puntos para construir la geometría del muro de la azotea

Eliminar reglas:
Recibimos una probabilidad de eliminar regla y una profundidad mínima que debe permanecer en la planta

    Repetir

        Si el valor aleatorio es menor que la probabilidad y el hijo está a mayor profundidad que la mínima indicada
            Eliminar regla

    Hasta i > número de hijos de la planta

Modificamos la forma de la planta creándola con las reglas no eliminadas

Crear forma de la planta:

Elegimos la forma principal de la planta en base a los parámetros recibidos
Creamos la geometría de puntos de la base
Si no hay que añadir hijos a la base de la planta
    Devolvemos los puntos que forman la geometría de la planta

Si hay que añadir hijos a la base de la planta

Repetir

Si las condiciones para añadir un hijo a la planta se cumplen
Si se indica en los parámetros de qué forma crear los hijos
Crear un conjunto de puntos para el hijo siguiendo las indicaciones de los parámetros
Si no se indica en los parámetros de qué forma crear los hijos
Crear un conjunto de puntos para el hijo de forma aleatoria pero consistente
Añadimos puntos que conforman una extensión de la planta al array de puntos de la planta

Hasta i > número de aristas de la planta

Eliminamos puntos duplicados en el array
Devolvemos los puntos que forman la geometría de la planta

Módulo GeometriaPlanta

Generar muros:

Calculamos el número de filas de textura tiene la planta en base a su altura
Repetir

Creamos un punto en la altura base de la planta y otro encima de él a la altura que debe tener la planta
Creamos caras geométricas con triángulos que recubren la sección de este vértice y el siguiente
Calculamos el número de columnas para repetir las texturas tileables en esta sección del vértice
Se añade a un array el conjunto de valores de numero de columnas y filas para saber cómo texturizar esta cara de la planta

Hasta i > número de vértices de la base

Generar azotea:
Si se ha indicado por parámetros generar azoteas
Obtenemos el conjunto de triángulos que conforman la geometría de la azotea
Repetir

Añadimos los tres vértices que constituyen el triángulo actual a un array para texturizar sobre cada uno de ellos

Hasta i > número de triángulos de la geometría de la azotea

Insertar ventanas:

Repetir

> Calculamos el número de ventanas que se pueden insertar en la cara actual
> Según la agrupación indicada por parámetros generamos los puntos de las ventanas de una forma u otra
> Rotamos la geometría formada por puntos para que las ventanas miren hacia afuera del edificio
> Fusionamos las geometrías de las ventanas de esta cara con la de las demás

Hasta i > número de vértices en la planta

Fusionar geometría de las ventanas con la geometría de la planta

Insertar elementos decorativos:

Repetir

> Mientras la posición elegida para el elemento no sea válida o se supere el tiempo límite para cada elemento
>
> > Calculamos las posiciones posibles del elemento y elegimos una
> > Calculamos las rotaciones posibles del elemento y elegimos una
> > Comprobamos si sus vértices no chocan con los de algún otro elemento
>
> Fusionar geometría del elemento con la geometría de la planta
> Actualizar zona ocupada por elementos

Hasta i > número de elementos a añadir